www.crs4.it/vic/

Keynote

Scalable Interactive Exploration of Massive Volumetric Data

Enrico Gobbetti

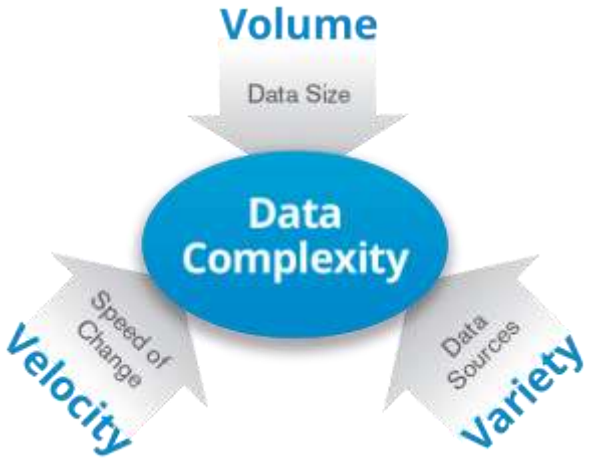
Director, CRS4/ViC

gobbetti@crs4.it

October 2015

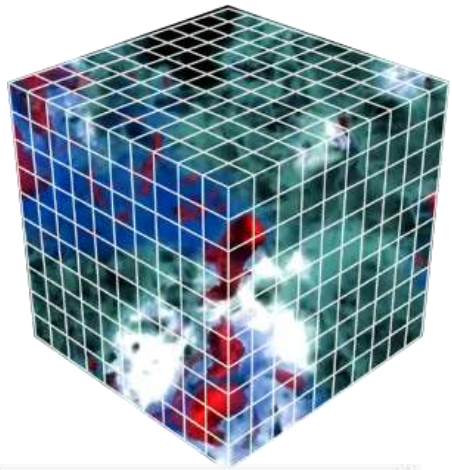
Big Data

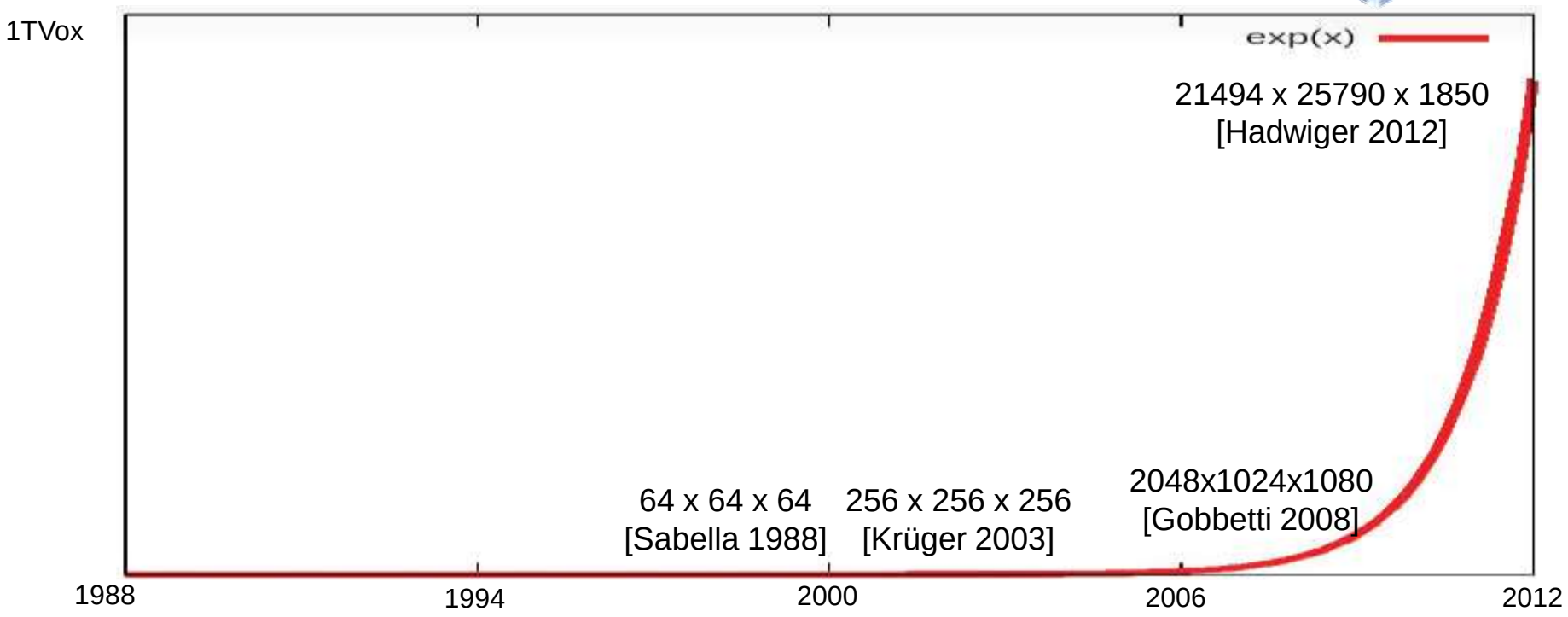
- **Explosion of data in all areas of science, engineering, health and business applications, driven by improvements in hardware and information processing technology**
  - **Acquisition:** 3D imaging, remote sensing, range scanners, massive picture collections, ubiquitous sensing devices, …
  - **Computing:** modeling, simulations…
- **Need for novel tools, techniques, and expertise!**
- **Our focus is 3D data**
  - Wide and deep impact on a variety of domains



In this talk: scalar volumes

- **Goal: interactively explore potentially unlimited volumetric datasets on single PCs**

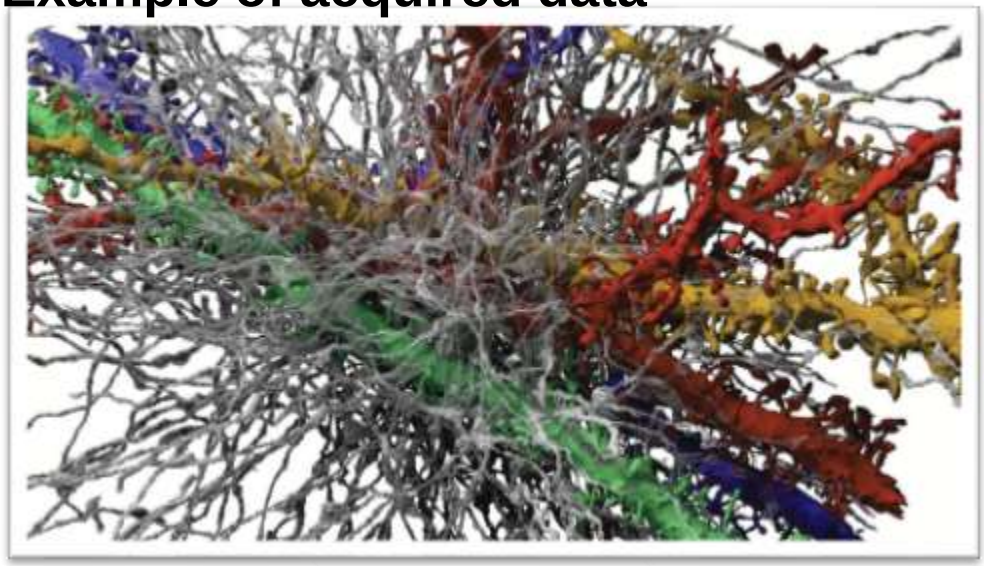




In this talk: scalar volumes

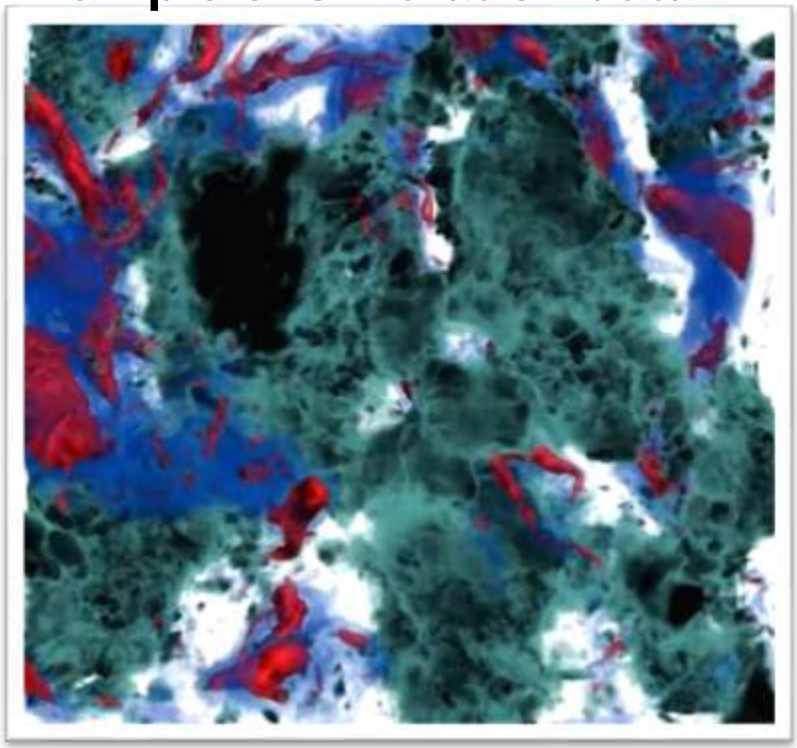
- **Goal: interactively explore potentially unlimited volumetric datasets on single PCs**

Example of acquired data



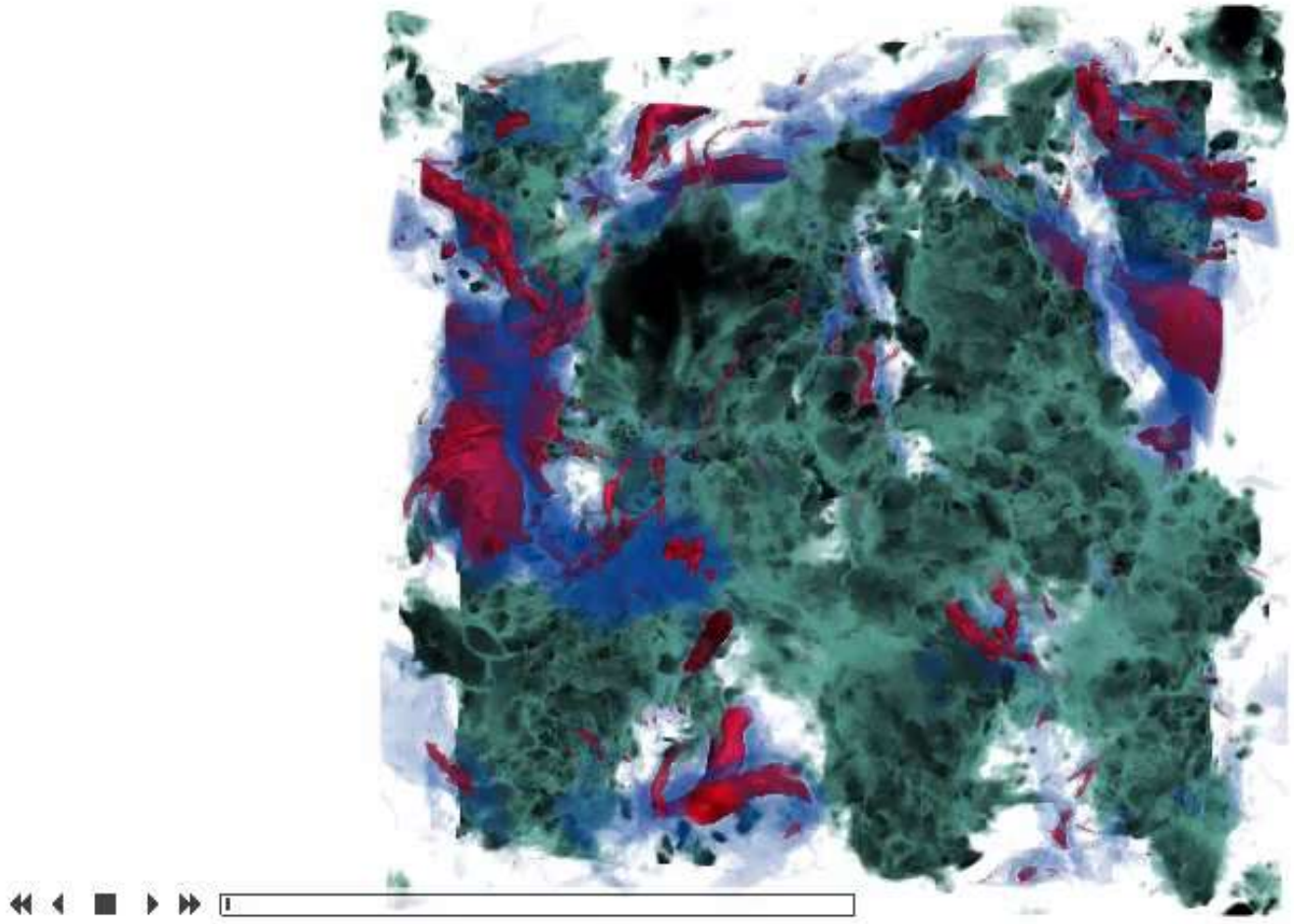
Connectome (Bobby Kasthuri, Harvard): ~60 μm^3 - **1 TeraVoxel**

Example of simulation data



JHU DNS Turbulence time varying simulation - 2048 time steps at 1024^3 – **2 TeraVoxels**

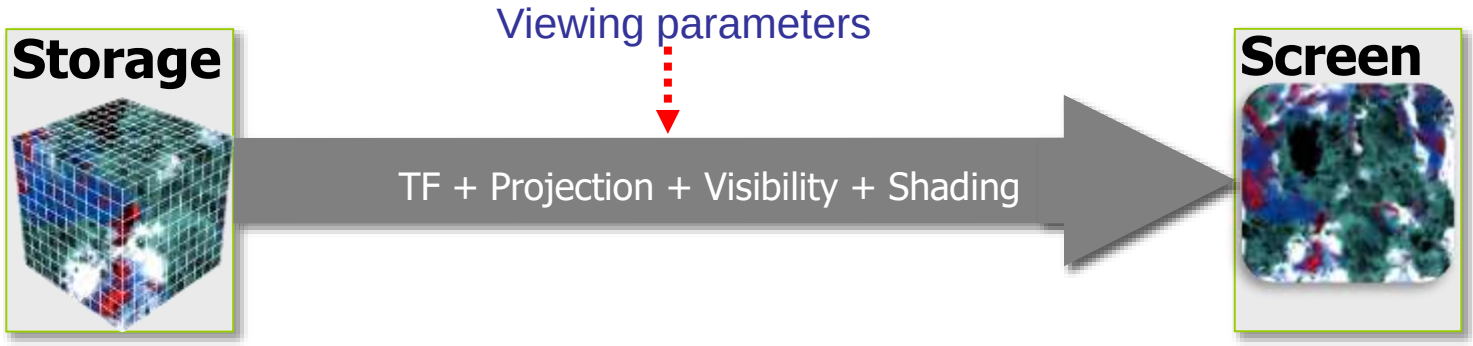
In this talk: scalar volumes



JHU Turbulence 1024^3 x 2048 frames float (8 TB) on Intel i7 16GB RAM with NVIDIA GeForce GTX 980

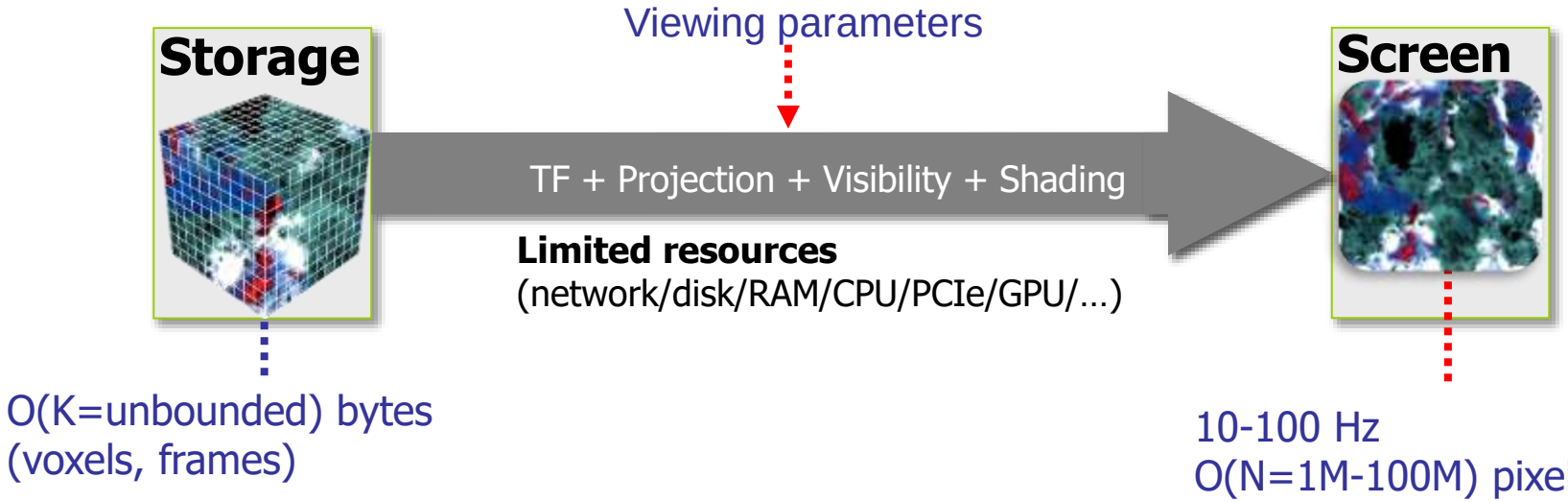
Massive volumetric data visualization

- **To explore massive 3D volumes we need to transform them at interactive into a synthetic image that can be displayed on the screen**
- **Two main families of algorithms…**
  - Raycasting algorithms
  - Rasterization-based algorithms (slicing)



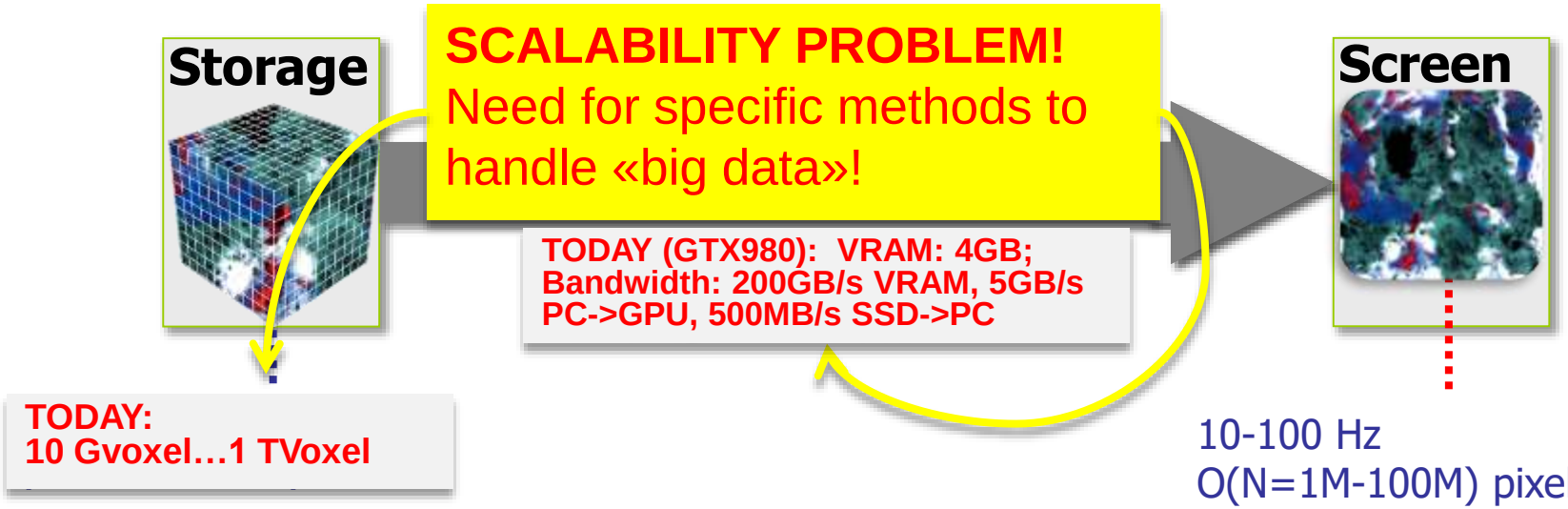
Massive volumetric data visualization

- **To explore massive 3D volumes we need to transform them at interactive into a synthetic image that can be displayed on the screen**
- **… large pressure on memory/bandwidth/comp.**
  - … basically traverse most voxels to produce images…
  - … sample each voxel multiple times (filtering, gradients…)



Massive volumetric data visualization

- **To explore massive 3D volumes we need to transform them at interactive into a synthetic image that can be displayed on the screen**
- **… large pressure on memory/bandwidth/comp.**
  - … basically traverse most voxels to produce images…
  - … sample each voxel multiple times (filtering, gradients…)



Massive volumetric data visualization

- **To explore massive 3D volume**
- **transform**

(voxels, frames)

10-100 Hz
O(N=1M-100M) pixels

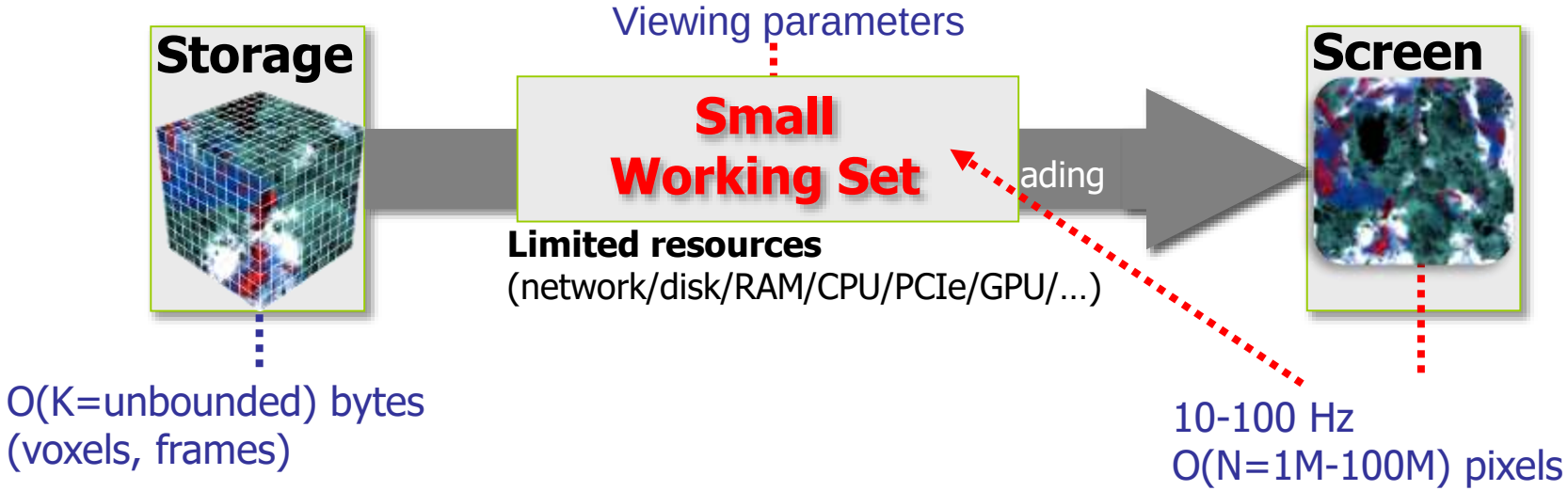
Scalability

- **Traditional HPC, parallel rendering definitions**
  - **Strong scaling** (more nodes are faster for same data)
  - **Weak scaling** (more nodes allow larger data)

- **Our interest/definition: output sensitivity**
  - **Running time/storage proportional to size of output instead of input**
    - Computational effort scales with visible data and screen resolution
    - Working set independent of original data size

Massive volumetric data visualization

- **Models of unbounded complexity on limited computers**
  - Need for **output-sensitive techniques** (O(N), not O(K))
    - We assume less data on screen (N) than in model (K $\rightarrow\infty$)
  - Need for **memory-efficient techniques** (low bandwidth and memory pressure, high caching)
  - Need for **parallel techniques** (high CPU/GPU core usage)



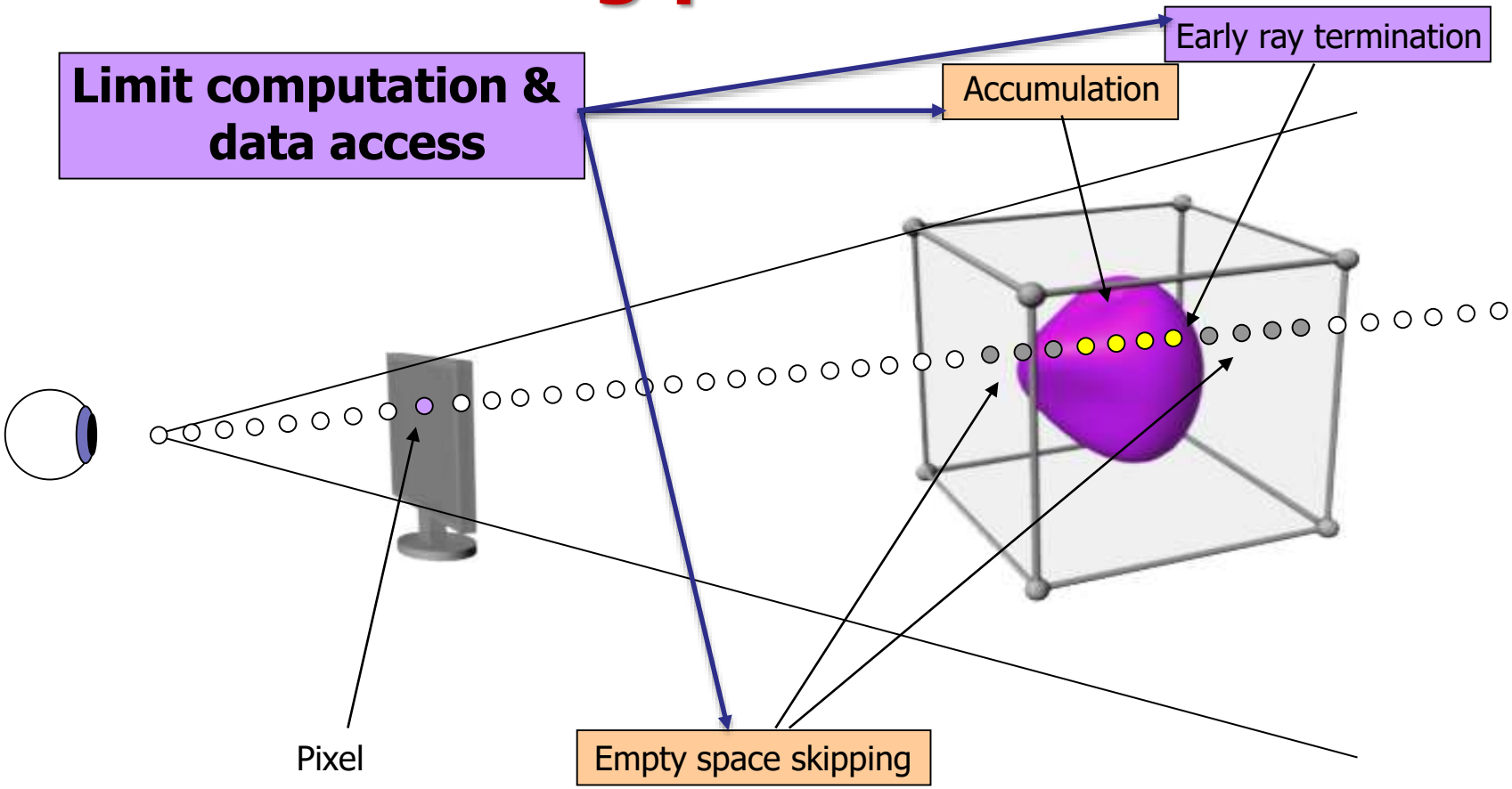
Solution strategies

- **Adaptive working set determination and rendering**
  - Multiresolution structures to partitition data and precompute prefiltered versions
  - Quick culling out of unneeded data to determine working set
  - Streaming/adaptive loading of working set data
  - Single pass vs. multipass rendering on working set
- **Compression and compression-domain rendering**
  - Reduce storage and bandwith
  - Compression-domain computations & deferred filtering

Adaptive working set determination and rendering

Ray-guided streaming with multiresolution GPU raycasting

Volume rendering problem

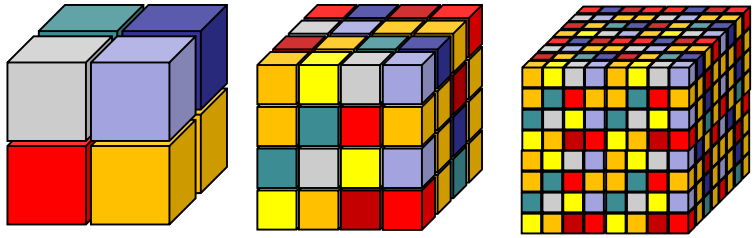


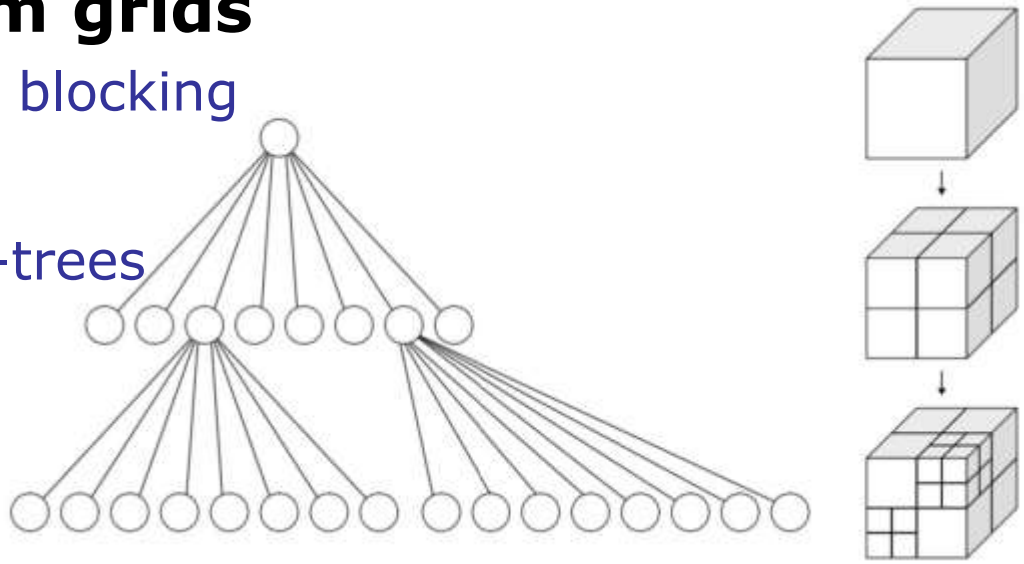
Order independent

Order dependent

Hierarchical Bricking

- **Prefiltering to represent data at multiple resolutions**
  - Essential for adaptivity
- **Pyramids of uniform grids**
  - Mipmaps, flat multires blocking
- **Tree structures**
  - **Octrees**, N^3 trees, kd-trees

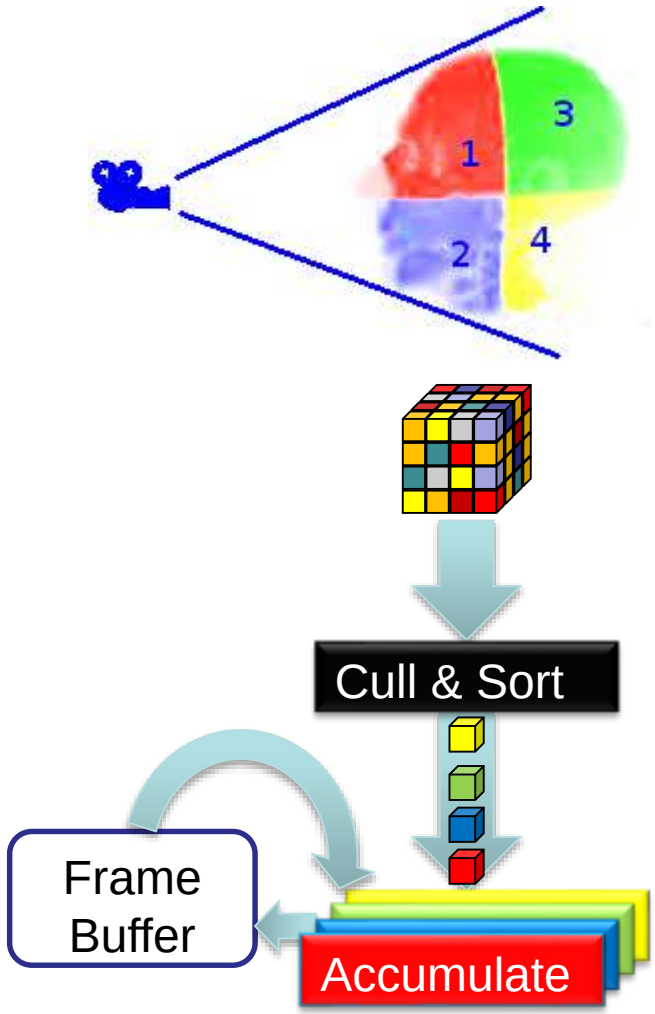




wikipedia.org

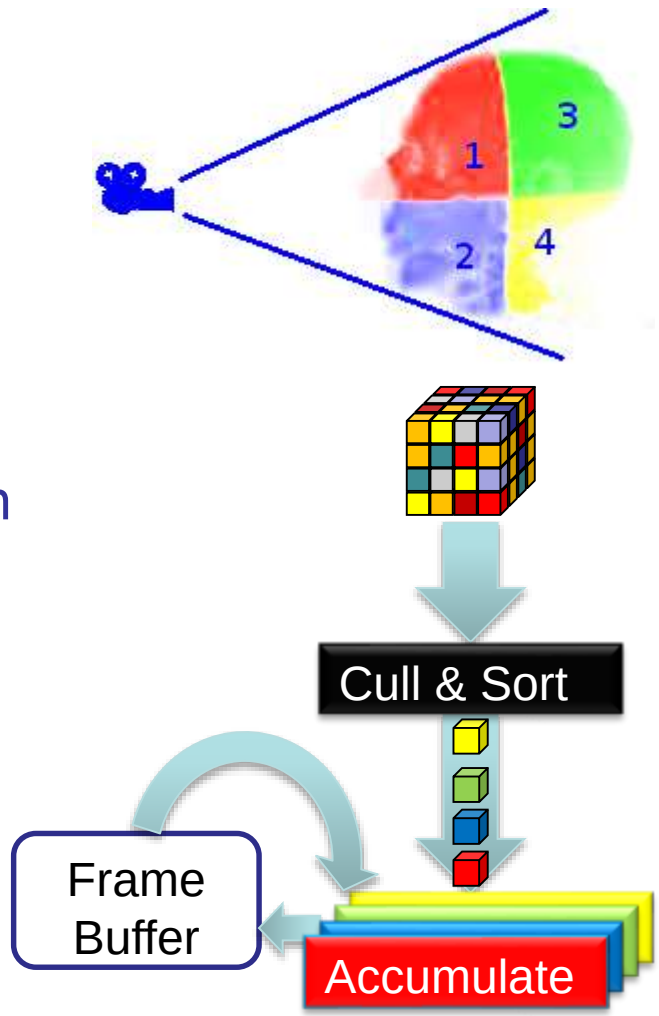
Out-of-core GPU Volume Rendering (Traditional)

- **Traditional out-of-core GPU volume rendering with large bricks (e.g., 256^3)**
  - **CPU working set determination prior to rendering**
    - global attribute-based culling (view-independent)
    - view frustum/occlusion culling (view-dependent)
  - **One rendering pass per brick**
    - Render back-to-front or front-to-back + frame buffer accumulation
  - **Sorting on CPU, can work in a streaming manner**



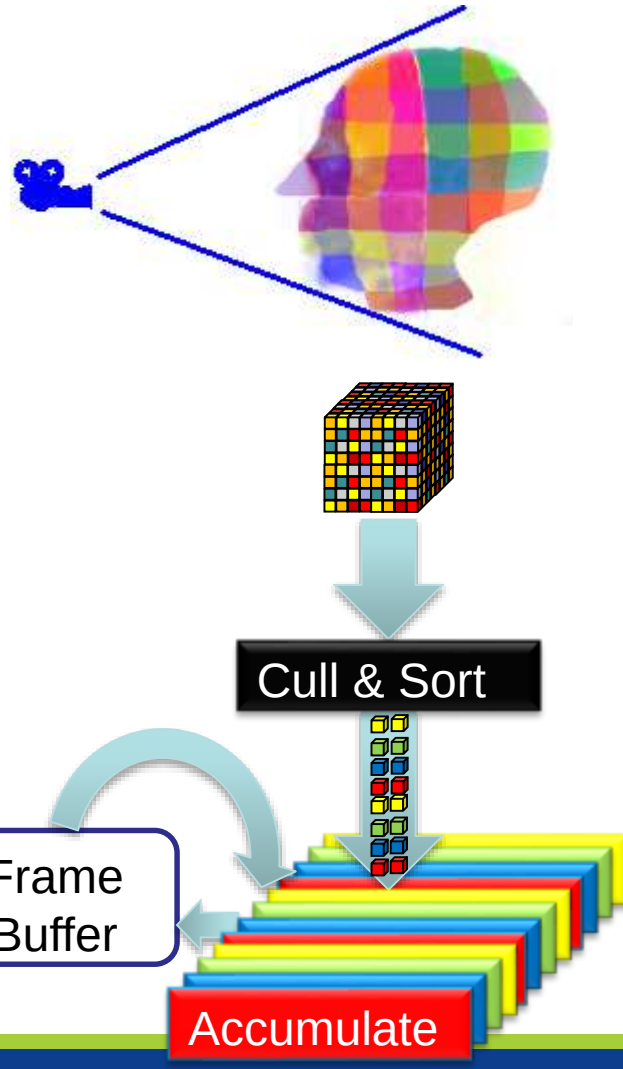
Out-of-core GPU Volume Rendering (Traditional)

- **Low culling efficiency and low flexibility**
  - Low granularity leads to inefficient bandwidth/memory usage
  - Hard to apply to indirect/curved rays (e.g., refraction, shadows, …)
  - need for frame buffer synchronization and auxiliary structures for efficient occlusion culling



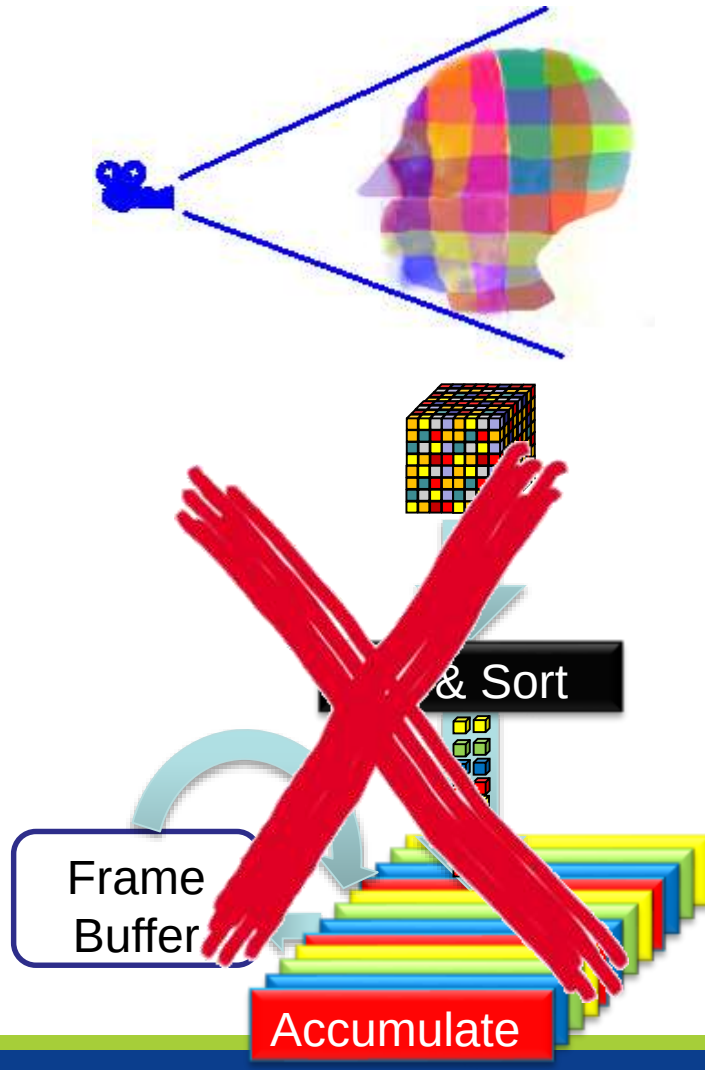
Out-of-core GPU Volume Rendering (Modern)

- **Modern out-of-core GPU volume rendering use small bricks (e.g., 16^3)**
  - **Better culling efficiency, tighter working set, but more bricks to cull and render…**
  - One pass per brick infeasible!
  - Working set determination infeasible on CPU



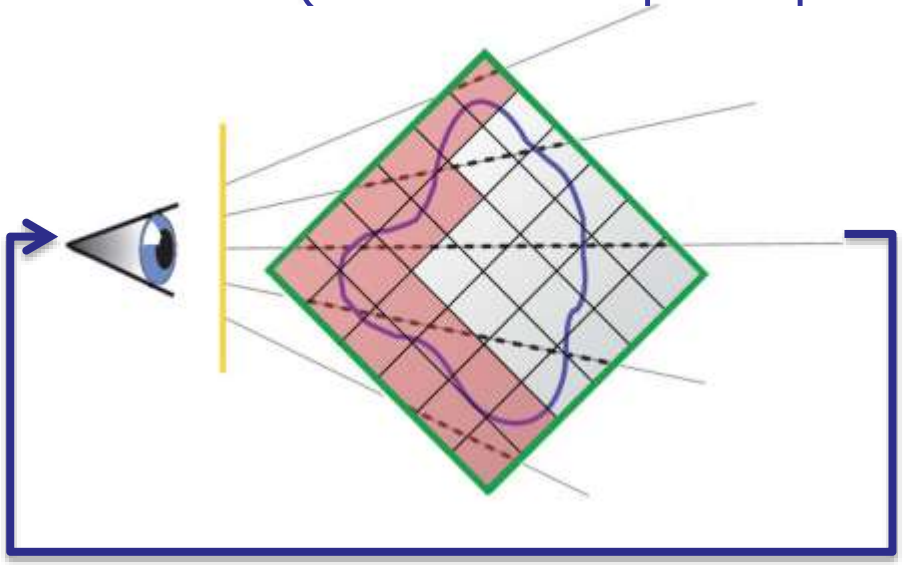
Out-of-core GPU Volume Rendering (Modern)

- **Modern out-of-core GPU volume rendering use small bricks (e.g., 16^3)**
  - **Working set determination on GPU**
  - **One-pass rendering on working set of bricks**
  - **Task-dependent brick size** (small for rendering, large for disk/network I/O)
    - See *Fogal et al., LDAV 2013* for benchmarks & analysis
- **Need for GPU algorithms + data structures!**



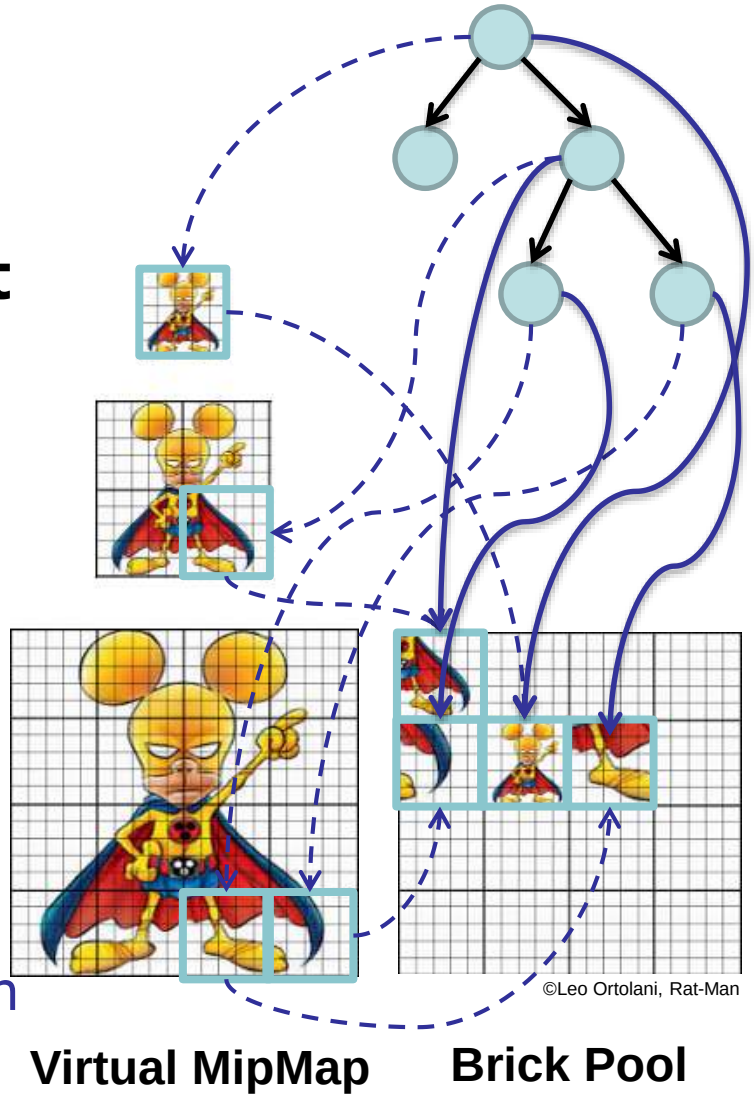
Working Set Determination (Modern)

- **Rays determine working set directly!**
  - Each ray writes out list of bricks it requires (intersects) front-to-back and the resolution it needs
  - Implicit view frustum culling (no extra step required)
  - Implicit occlusion culling (no extra steps or occlusion buffers)
  - Implicit LOD selection (no extra steps required)



Out-of-core GPU Volume Rendering (Modern)

- **GPU code must traversee a spatial/multires structure covering current working set**
  - E.g., octree
- **Brick data is shared in a common texture atlas**
  - Cache maintained using LRU/MRU policy
- **Spatial structure is maintained by the CPU and traversed by the GPU**
  - Octree performs address translation from virtual MipMap to brick pool



Ray-guided Architecture

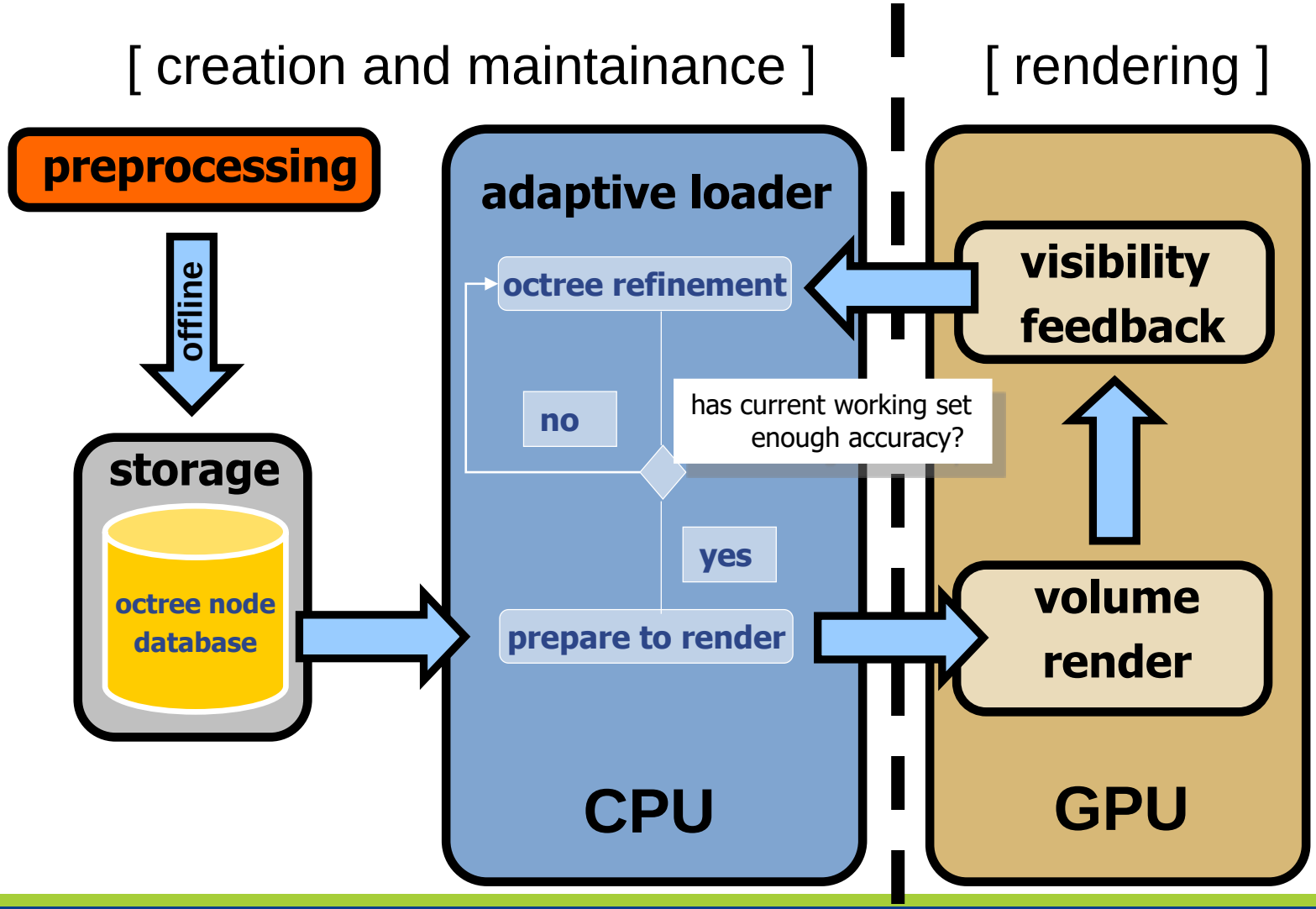
- **Early ray-guided streaming methods (Gobbetti 2008, Iglesias 2010):**
  - Multi-resolution out-of-core representation based on an octree of volume bricks
  - Adaptive CPU loading of the data from local/remote repository cooperates with separate render threads fully executed in the GPU
  - Stackless traversal of an adaptive working set
  - Exploitation of the visibility feedback
- **Many follow-ups by us and others**
  - E.g., **Crassin 2009**, ..., Gobbetti 2012, ..., Hadwiger 2014

E. Gobbetti, F. Marton, and J. A. Iglesias Guitián. **A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets**. *The Visual Computer*, 24, 2008.
J. A. Iglesias Guitián, E. Gobbetti and F. Marton **View-dependent exploration of massive volumetric models on large-scale light field displays**. *The Visual Computer*, 26, 2010.

Ray-guided Architecture

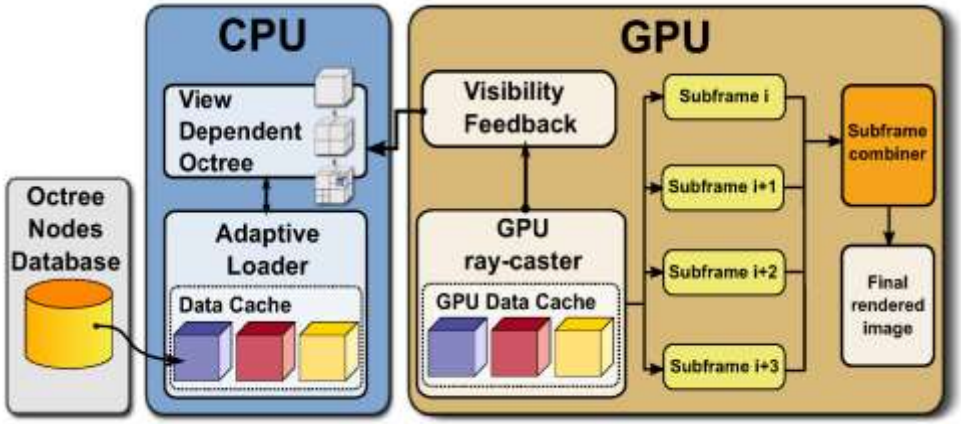
Year	Paper	Data size	Comments
2002	Guthe et al.	512 x 512 x 999 (500 MB) 2,048 x 1,216 x 1,877 (4.4 GB)	multi-pass, wavelet compression, streaming from disk
2003	Krüger & Westermann	256 x 256 x 256 (32 MB)	single-pass ray-casting
2005	Hadwiger et al.	576 x 352 x 1,536 (594 MB)	single-pass ray-casting (bricked)
2006	Ljung	512 x 512 x 628 (314 MB) 512 x 512 x 3396 (1.7 GB)	single-pass ray-casting, multi-resolution
2008	Gobbetti et al.	2,048 x 1,024 x 1,080 (4.2 GB)	**'ray-guided' ray-casting with occlusion queries**
2009	Crassin et al.	8,192 x 8,192 x 8,192 (512 GB)	**ray-guided ray-casting**
2010	Guitian et al.	4,096 x 4,096 x 4,096 (128 GB)	**ray-guided ray-casting**
2011	Engel et al.	8,192 x 8,192 x 16,384 (1 TB)	**ray-guided ray-casting**
2012	Hadwiger et al.	18,000 x 18,000 x 304 (92 GB) 21,494 x 25,790 x 1,850 (955 GB)	**ray-guided ray-casting visualization-driven system**
2013	Fogal et al.	8,192 x 8,192 x 8,192 (512 GB)	**ray-guided ray-casting**

Ray-guided Architecture

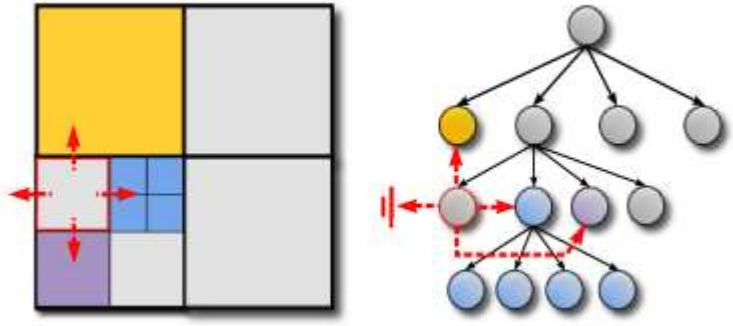


Ray-guided Architecture

- **Three GPU structures**
  - **Index tree**: octree structure of current working set
    - Octree with ropes in Gobbetti 2008, octree with pointers in Crassin 2009, Gobbetti 2010
  - **Brick cache**: data for current working set
    - Stored in texture cache, LRU policy
  - **Feedback buffer**
    - One entry per node in index tree



Architecture overview



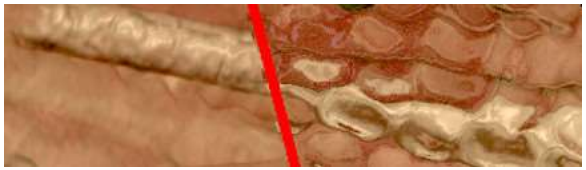
Neighbour pointer navigation

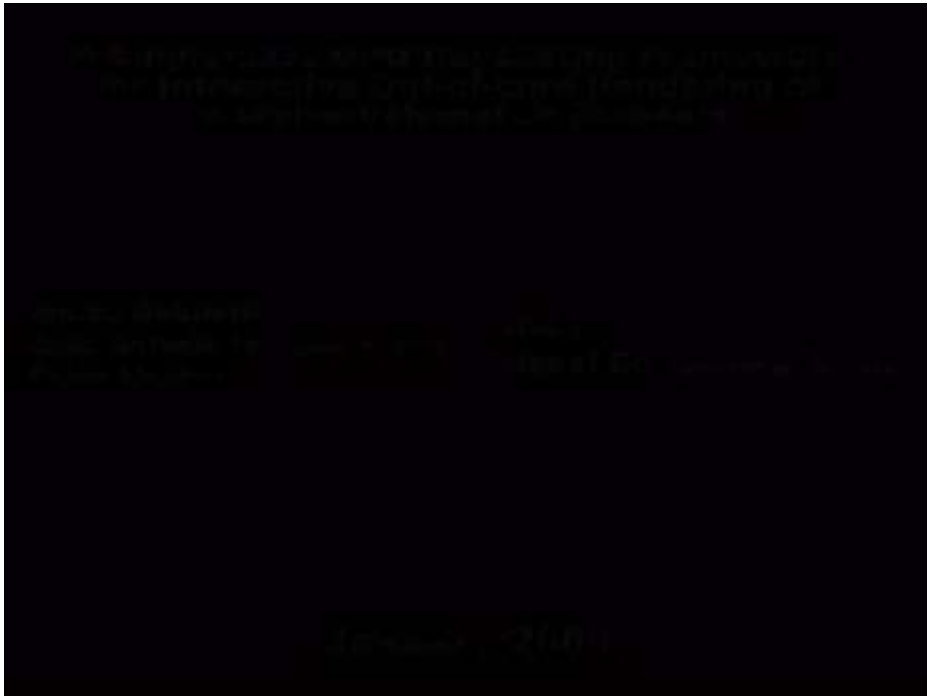
Ray-guided Streaming Algorithm

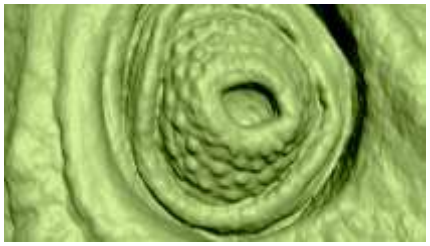
- **The adaptive loader maintains in-core a view-and-transfer function dependent cut of the out-of-core octree structure**
  - Uses it to update the GPU cache and Spatial Index.
  - Uses CUDA scatter write capability on a 8bit CUDA-array or modern OpenGL extensions (`GL_ARB_shader_storage_buffer_object`, …)
- **Basic principles:**
  - Update during rendering the **visibility status** of the nodes
  - **Refine** nodes marked as visible during the previous frame and considered inaccurate and non-empty according to the current transfer function
  - **Pull-up** visibillity data to inner nodes by recursive recombination and overly refined nodes
- **The cost amortized over full ray-casting is negligible**

Ray-guided Architecture

- **Scalable output-sensitive interactive technique**
  - Supports real-time out-of-core rendering of massive volumes on standard PC
  - Single-pass rendering with visibility culling applicable to a variety of situations
    - Transparency, refraction







Interactive exploration of a 16bit 2GB CT volume on a consumer NVidia 8800 GTS graphics board with 640MB (Gobbetti et al., 2008)

Ray-guided Architecture

Interactive exploration of a 8GVoxel Dataset on 35MPixel Light Field Display (Iglesias Guitian et al., TVC, 2010)

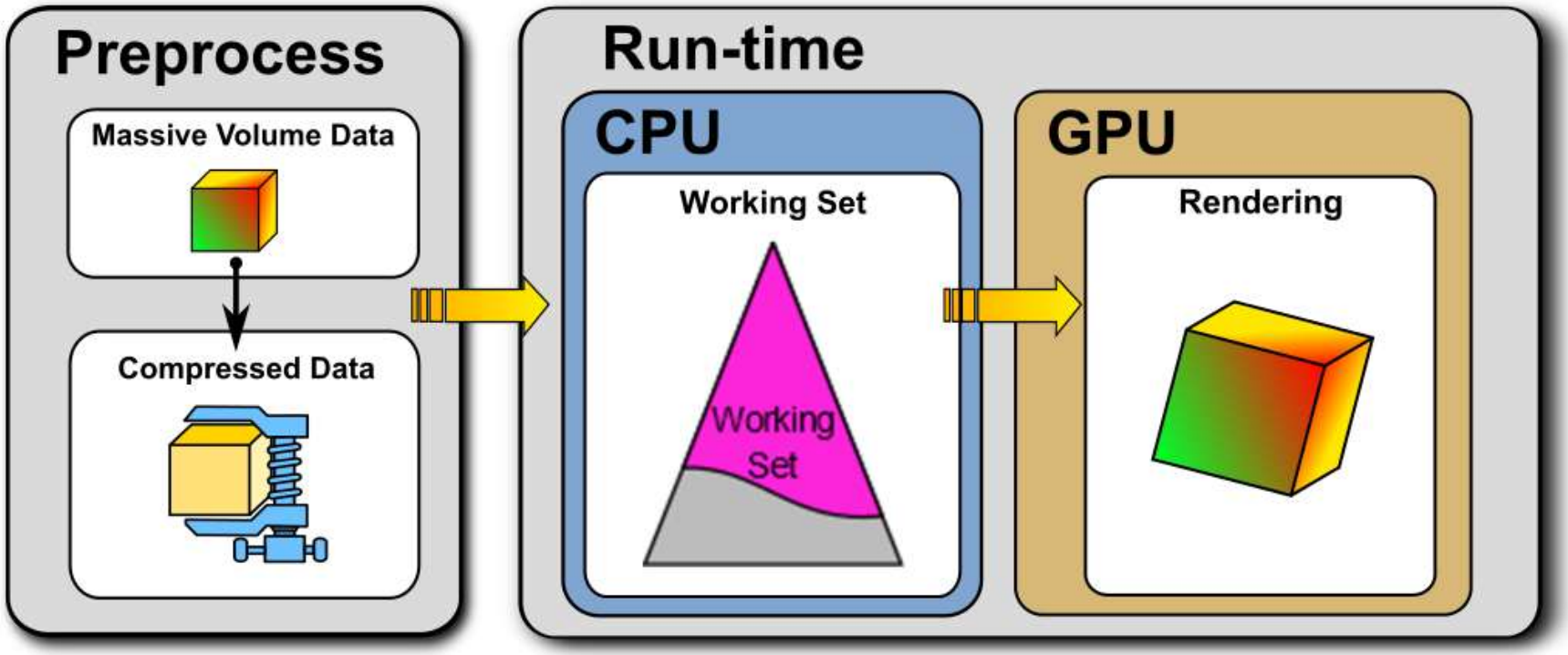
Optimizing memory & bandwidth

COMPRESSION-DOMAIN RENDERING

Optimizing memory & bandwidth

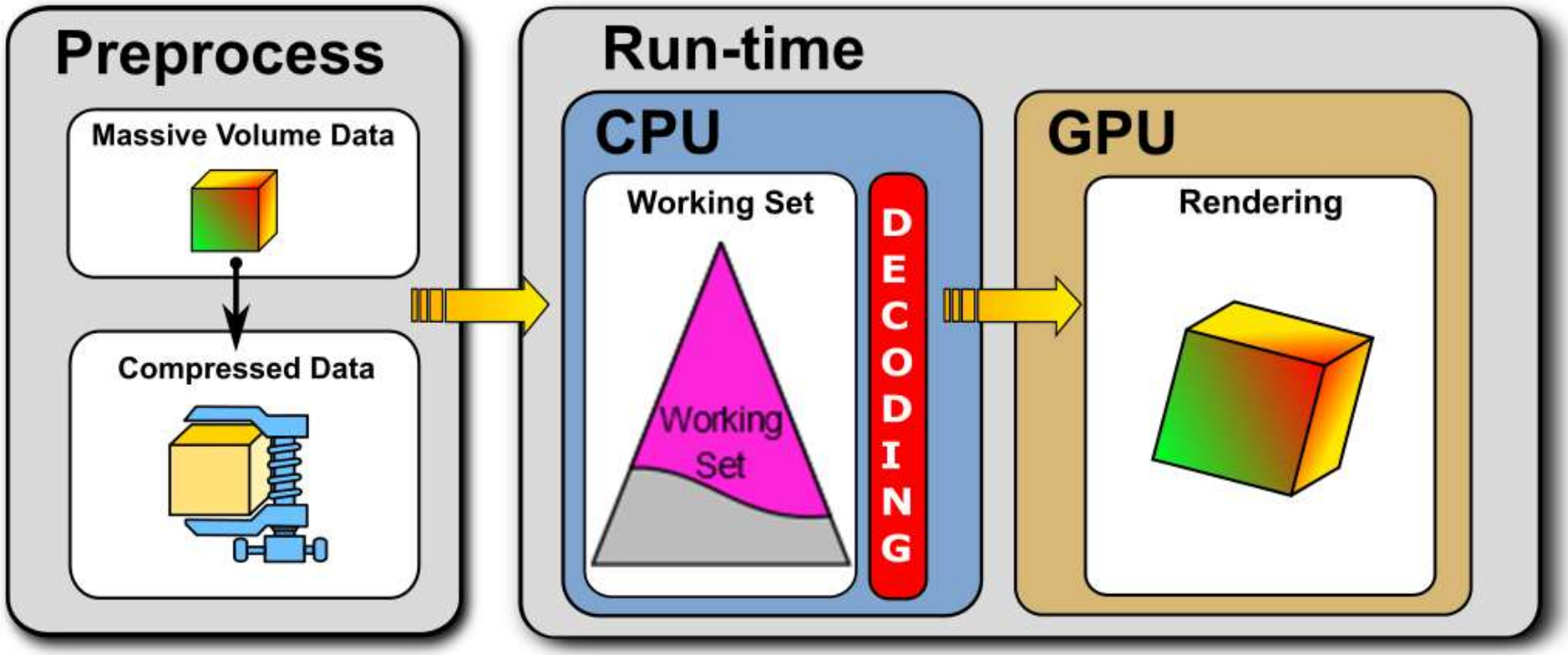
- **Long data transfer times and GPU memory size limit maximum update rate and dataset size**
- **Must combine compression with LODs and adaptive loading**
  - For maximum benefits, data must travel in compressed format through all the pipe-line
- **We can afford "slow" offline compression and precomputation, but we require fast real-time data decoding, interpolation and shading**
  - Spatially independent random-access to data
- **=> New architecture, and new compressed data formats!**

Architecture



- **Typical architecture for massive datasets**

Architecture



- **Decompression on CPU before rendering**

Decompression on CPU before Rendering

- **Decompression (generally lossless) of data streams**
  - E.g. Wavelet-based time-space partitioning [Shen et al. 2006]

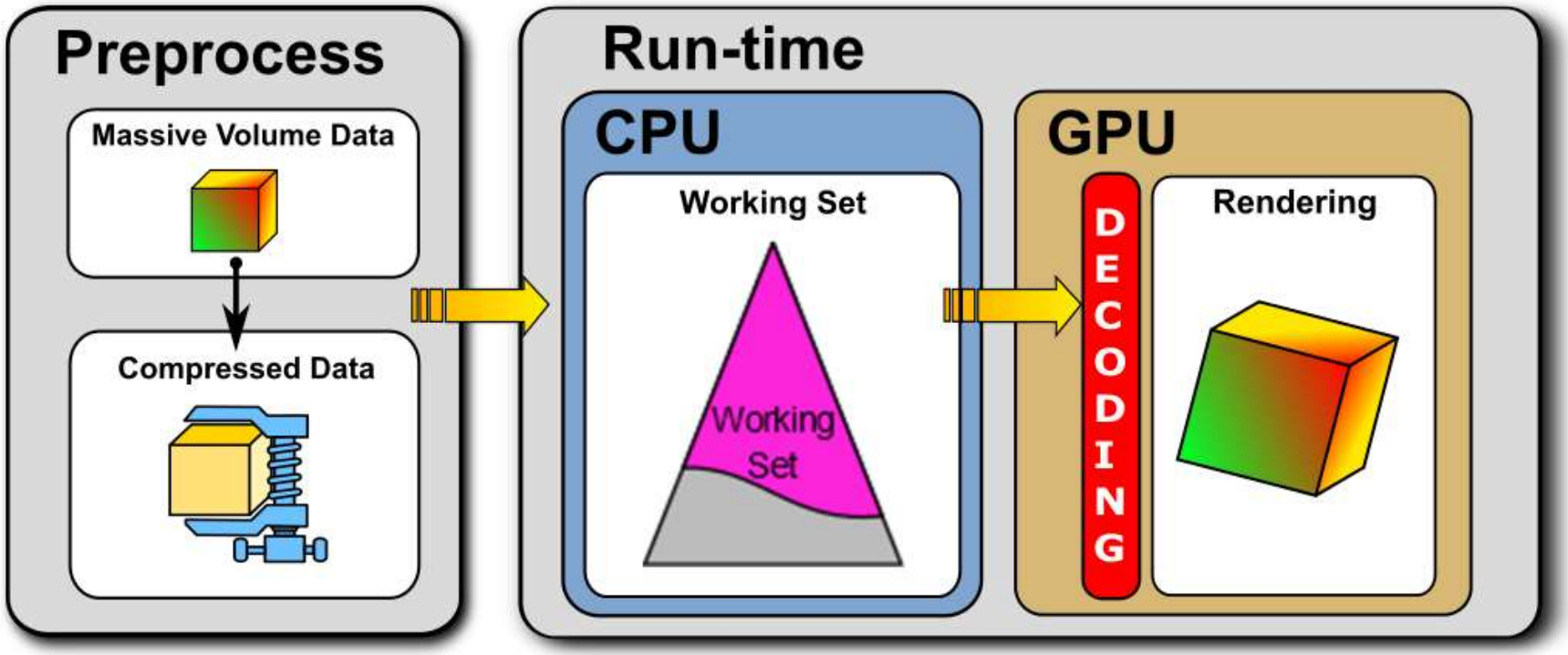
- **☺ Reduces the storage needs**
- **☺ Support faster and remote access to data**

Decompression on CPU before Rendering

- **Decompression (generally lossless) of data streams**
  - E.g. Wavelet-based time-space partitioning [Shen et al. 2006]

- **☹ Large memory resources needed**
- **☹ Inefficient use of the GPU memory bandwidth**
- **☹ CPU decompression is generally not sufficient for time-varying datasets**
  - Common in hybrid architectures [Nagayasu'08, Mensmann'10]

Architecture

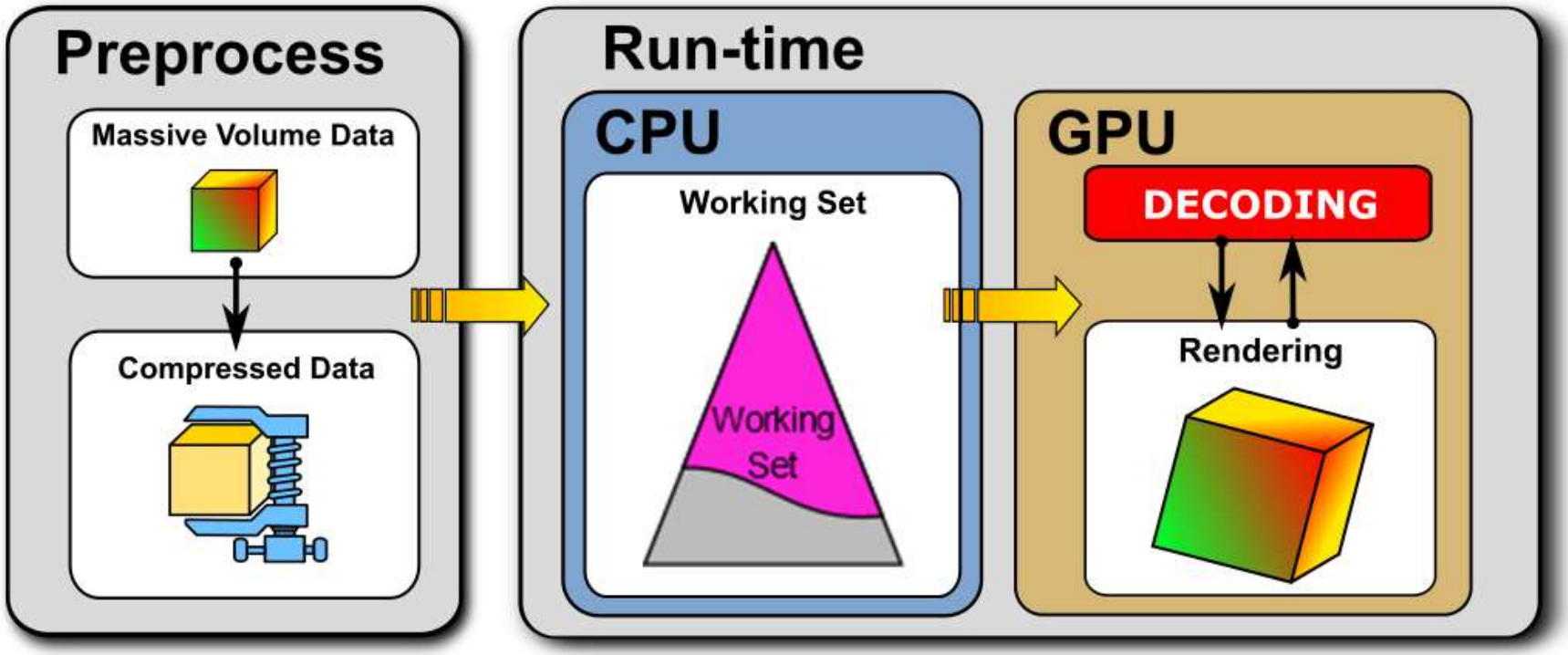


- **Full decompression on GPU before rendering**

Full Decompression on GPU before Rendering

- ☺ **Speed-up decompression by using the GPU**
  - Full working set decompression
    - [Wetekam'05, Mensmann'10, Suter'11]
- ☺ **Works for time-varying data**
  - … but each single time step should fit in the GPU memory and has to be decoded fast enough
- ☹ **Limit the maximum working set size …**
  - … to its size in uncompressed format

Architecture



- **Transient local decompression during rendering**

Decompression during Rendering

- **Transient and local decoding occurs on-demand during the rendering process**
  - The entire dataset is never fully decompressed
  - Data travels in compressed format from the CPU to the GPU
- **☺ Make the most of the available memory bandwidth**
- **On-demand, fast and spatially independent decompression on the GPU:**
  - Pure random-access
  - Local and transient decoding

Pure random-access

- **Data decompression is performed each time the renderer needs to access a single voxel**
  - **Hardware-supported formats**
    - Fixed-rate block-coding methods
      - E.g. OpenGL VTC [Craighead'04], ASTC [Nystad'12]
    - GPU implementations of per-block scalar quantization
      - [Yela'08, Iglesias'10]
  - **Fast, full support for filtering**
  - **Limited choice, not scalable, low compression performance**

Pure random-access

- **Data decompression is performed each time the renderer needs to access a single voxel**
  - **Per-voxel decoding**
    - GPU implementations of VQ and HVQ
      - [Schneider'03, Fout'07]
    - ☹ Costly in performance
    - ☹ Rarely implement multisampling or HQ shading
      - For this they rely on deferred filtering architectures
  - **Slow, no support for filtering (typically single-sample nearest neighbor)**
  - **Limited choice, not scalable, low compression performance**

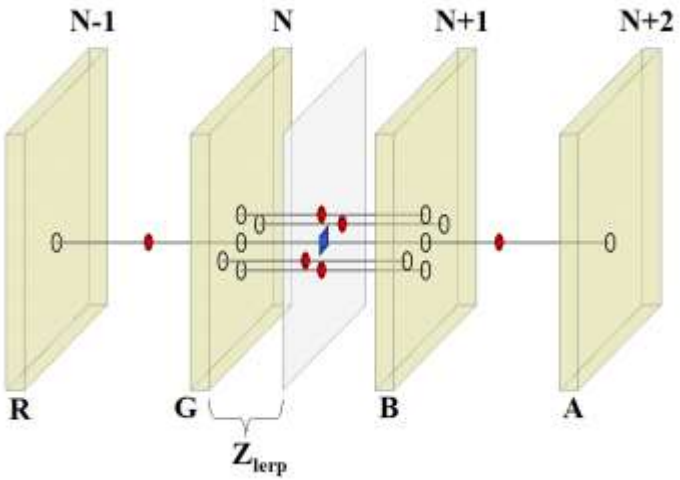
Local and transient decoding

- **Partial working set decompression**
- **Interleaving of decoding and rendering**
- **☺ Exploit better the GPU memory resources**
- **☺ More choice of compression methods**

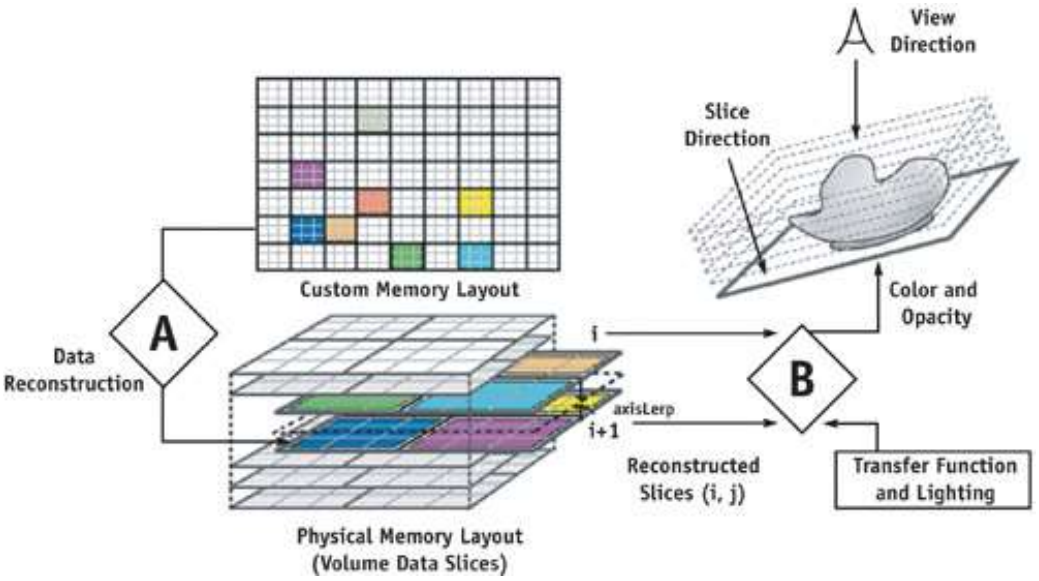
Deferred filtering

- **Deferred filtering solutions [Fout'05, Wang'10]**
  - Decompress groups of slices into textures, which can be reused (multisampling, filtering) during rendering for gradient and shading computations
  - Exploit spatial coherence and supports high-quality shading
  - Proposed as a single resolution approach

Deferred filtering (single resolution)



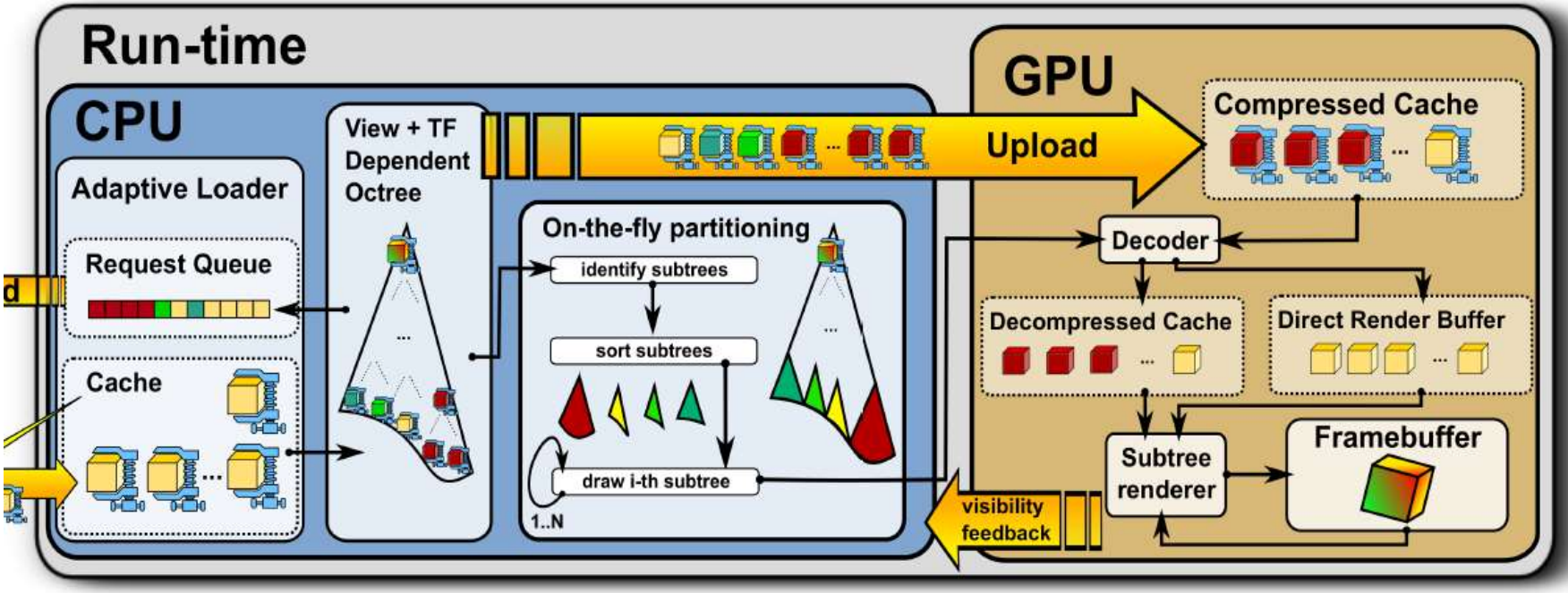
Gradient computation using central differences [Fout'05]



Rendering from a Custom Memory Format Using Deferred Filtering [Kniss'05]

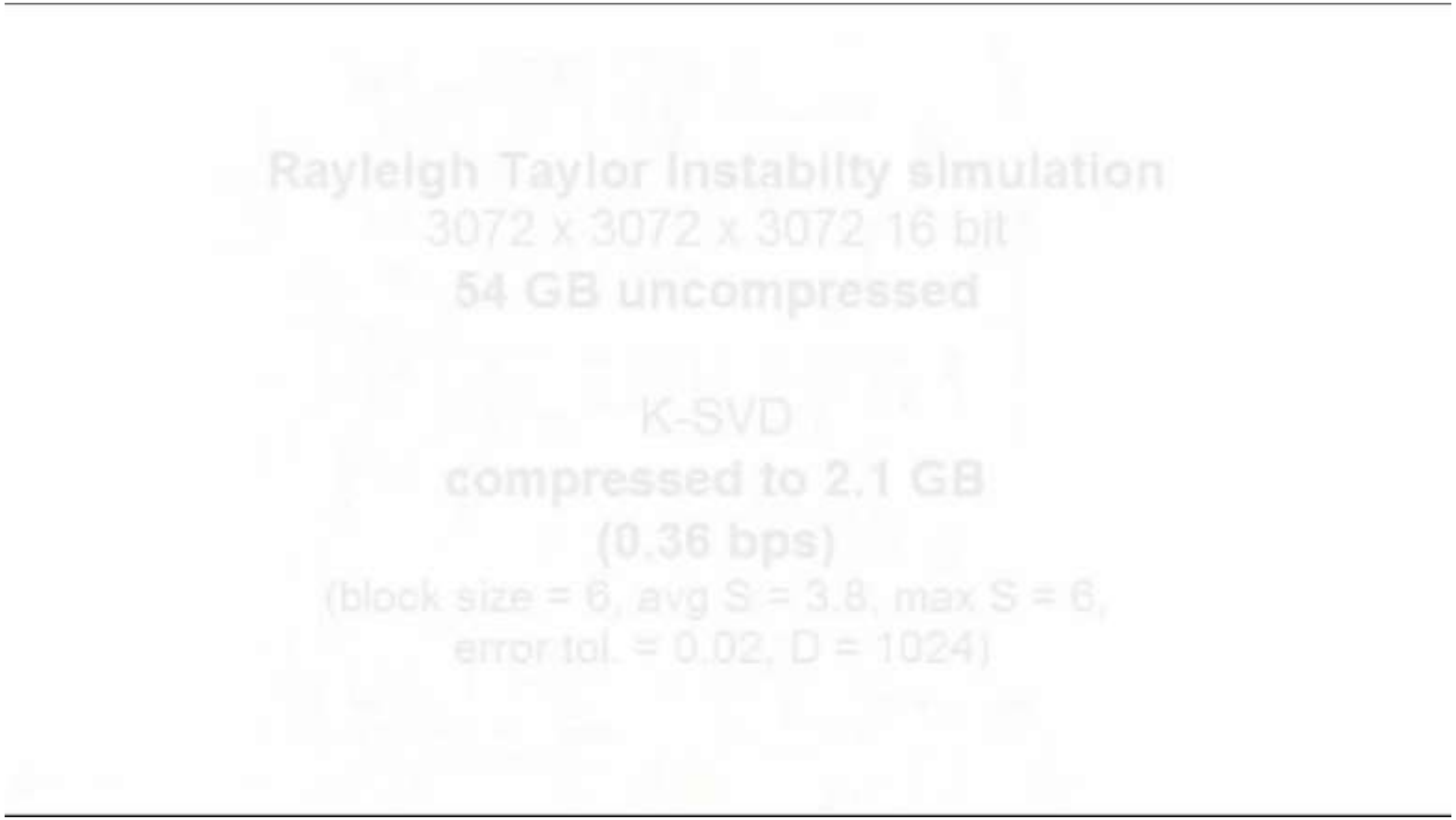
Deferred filtering (multiresolution)

- **Extended deferred filtering to work with multiresolution data structures [Gobbetti'12]**
  - Frame-buffer combination of partial portions of an octree



Deferred Filtering of an octree structure in GPU [Gobbetti'12]

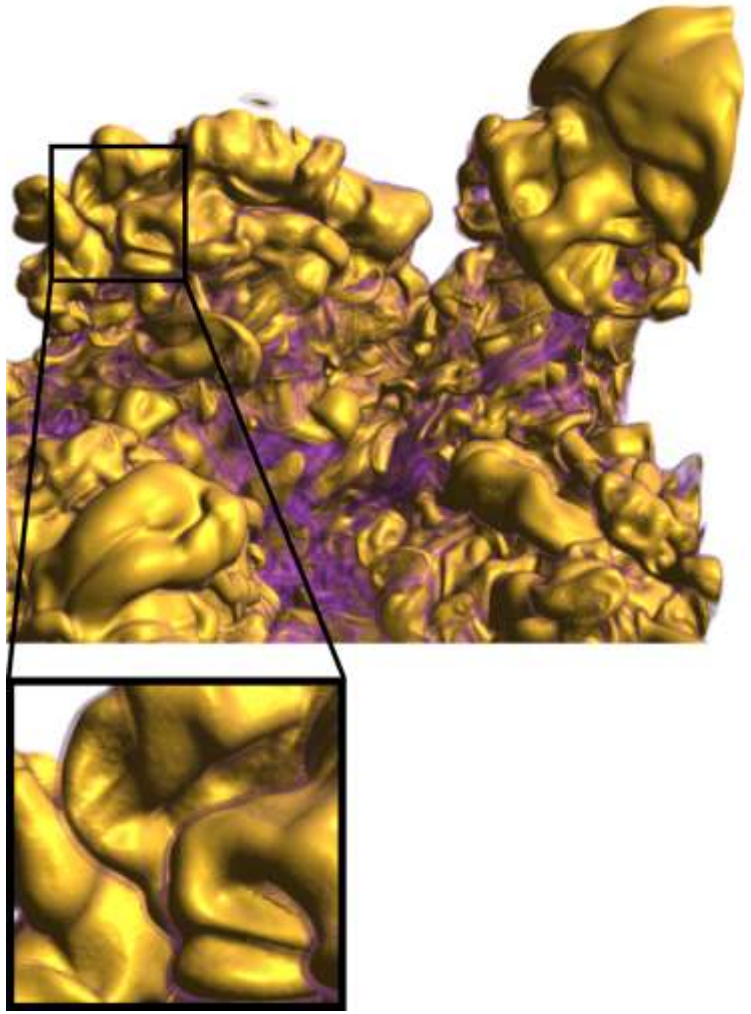
Multiresolution deferred filtering



Rayleigh-Taylor 3072^3 16bit (54 GB) on Intel i7 16GB RAM with NVIDIA GeForce GTX 980

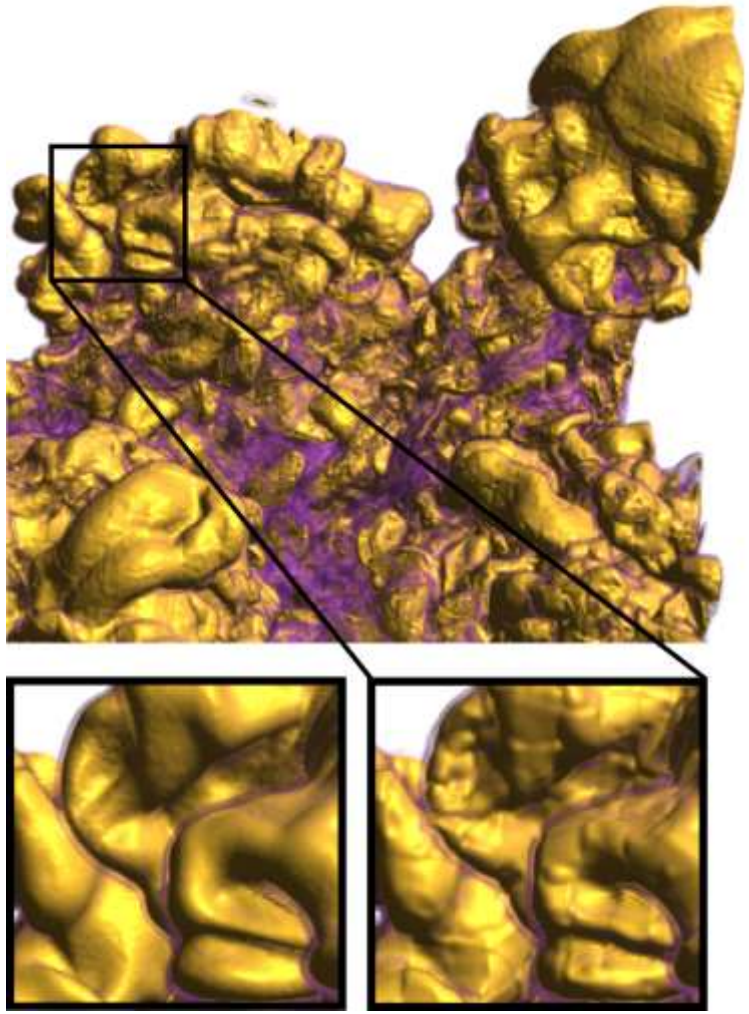
Blocking artefacts in compressed rendering

- **High compression ratios may produce artifacts between adjacent blocks**
  - Blocks are individually compressed and do not match at boundaries
  - Small differences create gradients



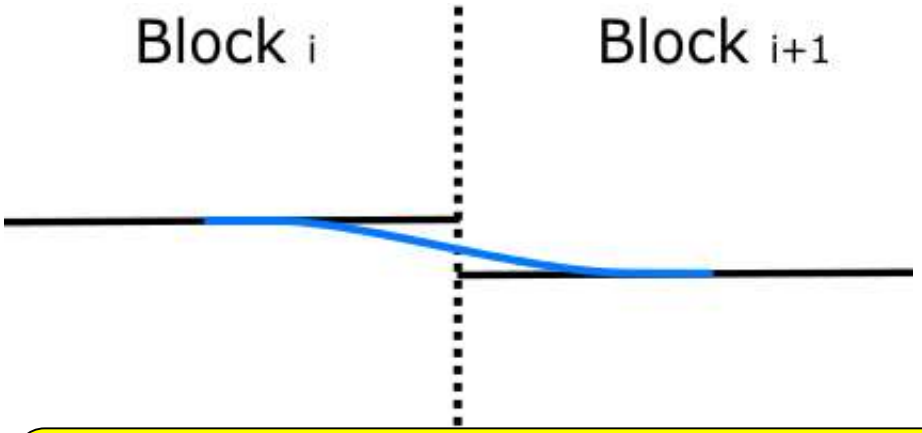
Blocking artefacts in compressed rendering

- **High compression ratios may produce artifacts between adjacent blocks**
  - Blocks are individually compressed and do not match at boundaries
  - Small differences create gradients



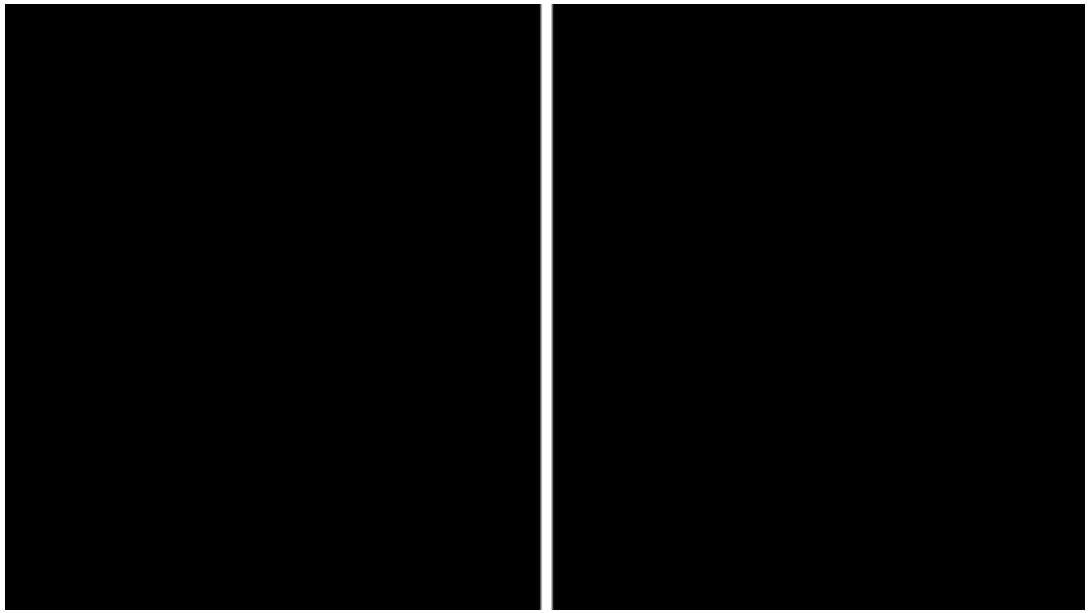
Deblocking

- **Filter decompressed data before rendering**
  - Smooth transition across boundaries
  - Good balance artifact removal / feature preservation
  - Low impact on performance



F. Marton, J. A. Iglesias Guitián, Jose Diaz, and E. Gobbetti.
Real-time deblocked GPU rendering of compressed volumes.
Proc. VMV. Pages 167-174, October 2014.

Deblocking results



Johns Hopkins Turbulence simulation (512 time steps)

512 x 512 x 512 32 bit

256 GB uncompressed

K-SVD [Gobbetti et al. 10]

Compressed to 1.6 GB

(0.2 bps)

(block size = 8, K = 4, D = 1024)

Architecture Wrap-up

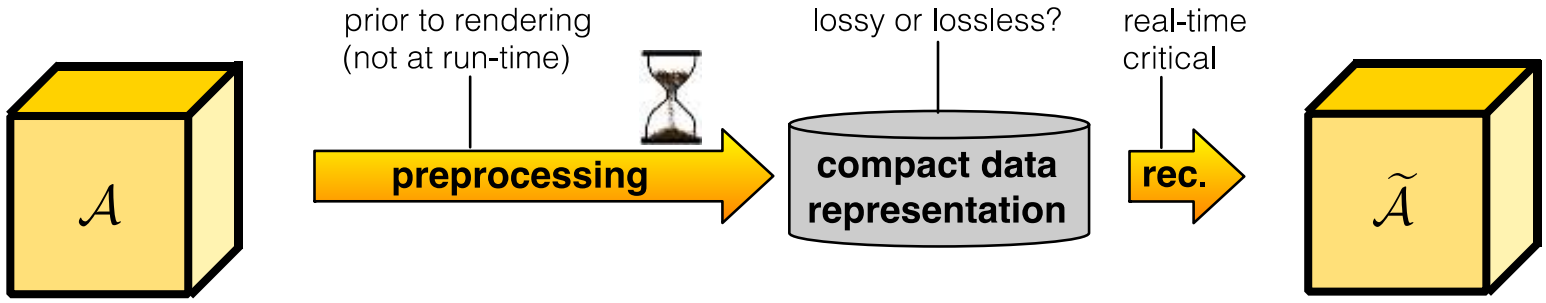
- **Local and transient reconstruction is a key factor in compression domain volume rendering**
  - Requires design at different levels
    - Compact data model
    - Rendering architectures
- **Pure voxel-based random-access and online/transient decoders avoid the full working set decompression**
- **Efficient decompression is specially important in lighter clients**
  - e.g. mobile, constrained network speeds

Optimizing memory & bandwidth

COMPACT REPRESENTATIONS

Compact representations

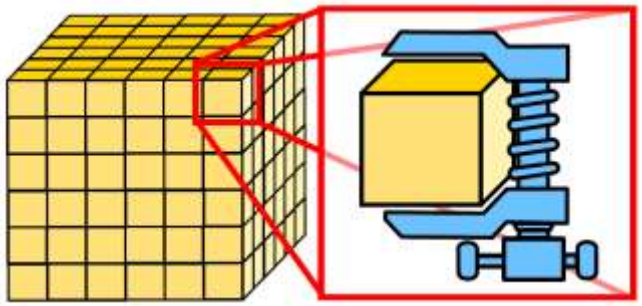
- **The presented architecture supports many brick-based compression methods with fast GPU decompression**



- **Off-line: Achieve high compression ratio with high quality**
- **On-line: Reconstruct in real-time**

Compact representations

- **Decompose the volume into a multiresolution octree structure, where each octree node (brick) is decomposed in smaller blocks. Each block is compressed**

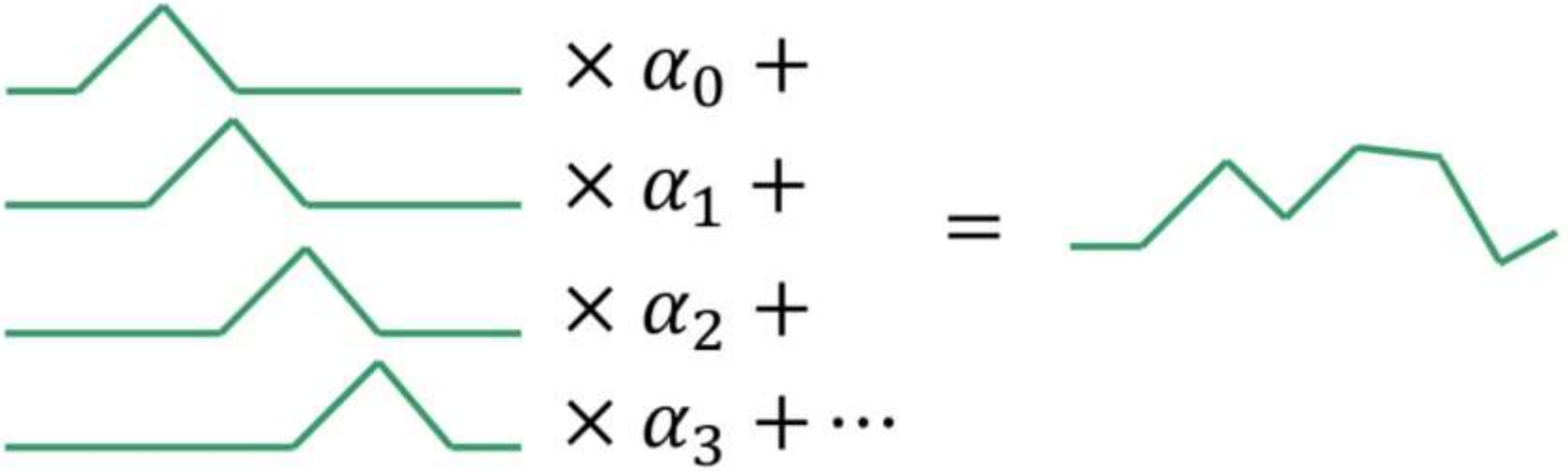


Single octree node containing overlapping information

- **Block is 3D signal that must be compressed/decompressed**

Representing signals

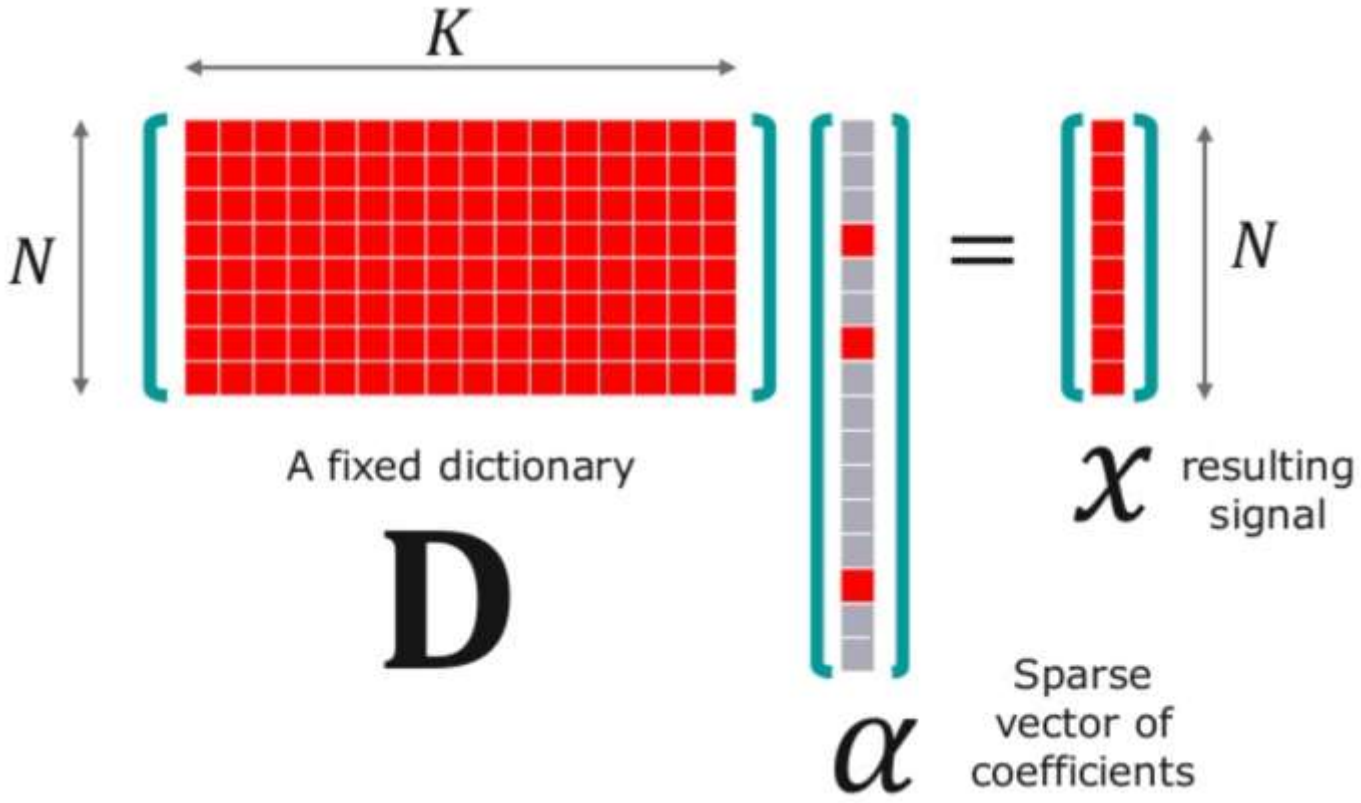
- **Signals can be represented as linear combinations of something we already know (the *basis*)**



Drawing courtesy of Manny Ko

The sparse signal model

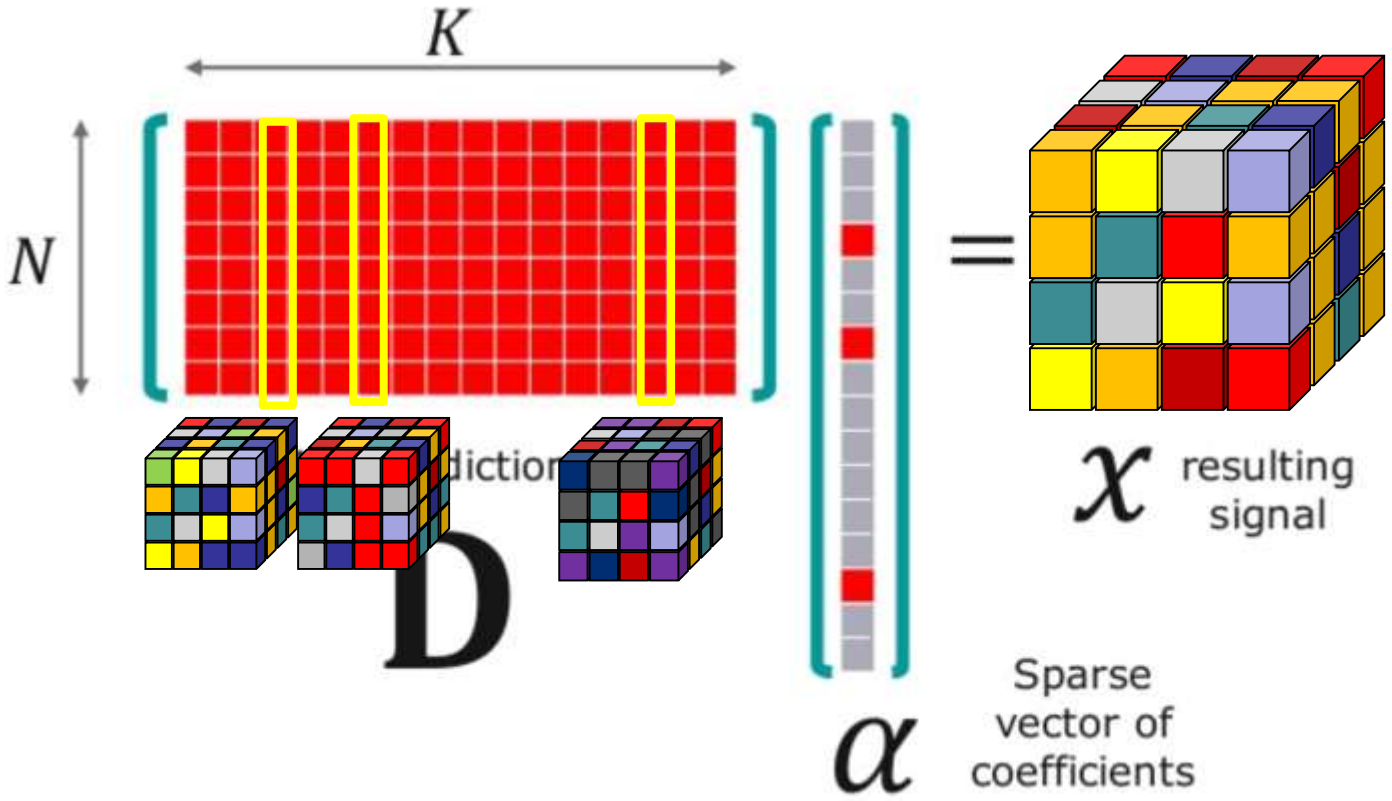
- **Compression achieved by selection, truncation and quantization of coefficients**



Drawing courtesy of Manny Ko

The sparse signal model

- **Compression achieved by truncation and quantization of coefficients**



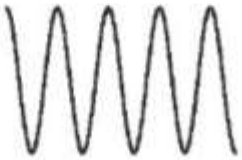
Predefined Orthonormal Bases

- **The most classic choice is to use fixed and orthonormal bases to define the dictionary D (K=N , $\int_{-\infty}^{+\infty} b_i(t) b_j(t)\, dt = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$)**

Common ONBs

- Fourier Basis

$b_k(t) = e^{i2pkt}$



- Gabor Functions

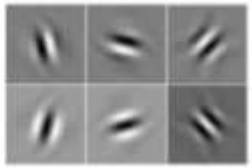
$b_{k,n}(t) = \omega(t - bn)e^{i2pkt}$

- Wavelets

$b_{m,n}(t) = a^{-m/2} x(a^{-m}t - bm)$

- Contourlet

$b_{j,k,\mathbf{n}}(t) = \lambda_{j,k}\left(t - 2^{j-1}\mathbf{S}_k\mathbf{n}\right)$



Drawing courtesy of Manny Ko

Predefined Orthonormal Bases

- **Predefined orthonormal bases have many advantages…**
  - The dictionary is implicit (analytic formulation)
  - Mathematical properties are well studied
  - Often fast algorithms for projection and reconstruction
- **Lots of applications in volume rendering**
  - discrete Fourier transform (1990;1993)
  - discrete Hartley transform (1993)
  - discrete cosine transform (1995)
  - discrete wavelet transform (1993)
  - laplacian pyramid/transform (1995;2003)
  - Burrows-Wheeler transform (2007)

Predefined Orthonormal Bases

- **Predefined orthonormal bases have many limitations…**
  - Optimal only for specific synthetic signals
  - Limited expressiveness, all signals behave the same
  - Real-world signal often require lots of coefficients
    - Truncation leads to aliasing
- **Interest is now shifting towards learned bases**
  - Data-dependent dictionary
  - Overcomplete basis (K>N), not orthonormal
    - There is more than one way to represent a signal
    - By relaxing ONB rules we can better represent signals using less coefficients

Learned Bases

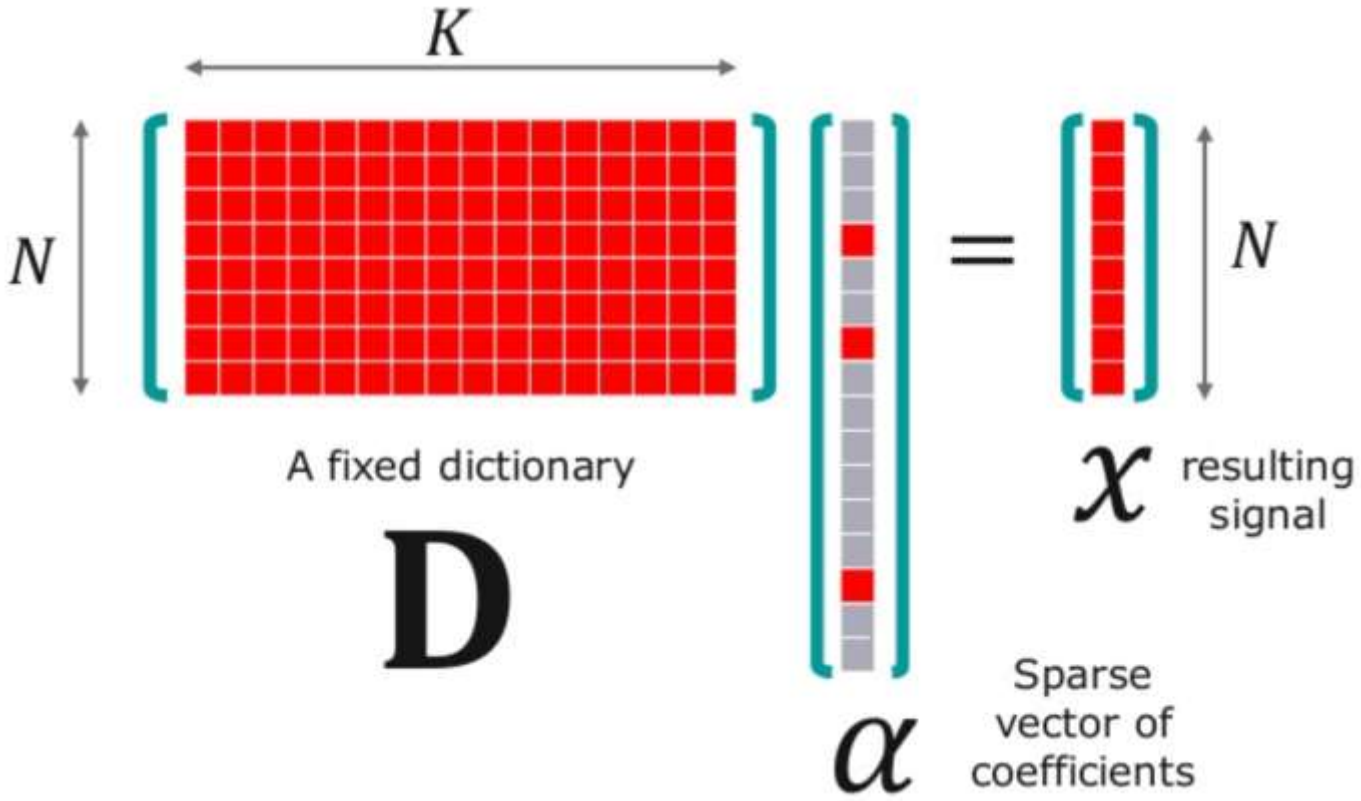
- **Early work based on vector quantization**
  - Special case where sparsity is forced to one
  - Many applications in volume rendering (1993;2003)
  - Achievable quality dependent on dictionary size
- **Recent work focused on more scalable solutions**
  - Karhunen-Loeve transform + VQ (2007)
  - **Tensor approximation (2010; 2011) (UZH+CRS4)**
  - **Sparse coding (2012; 2014) (CRS4)**

Learned Bases

- **Early work based on vector quantization**
  - Special case where sparsity is forced to one
  - Many applications in volume rendering (1993;2003)
  - Achievable quality dependent on dictionary size
- **Recent work focused on more scalable solutions**
  - Karhunen-Loeve transform + VQ (2007)
  - **Tensor approximation (2010; 2011) (UZH+CRS4)**
  - **Sparse coding (2012; 2014) (CRS4)**

Sparse coding

- **Explicit representation of dictionary D, which is learned from the dataset prior to encoding**



Sparse coding of volume blocks

- **Two phases**
  - Learn dictionary **D** from data
  - Encode data using computed **D**
    - Compression is achieved by storing indices and magnitudes

$$\sum_{i=1}^{S} c_i * \mathbf{d}[a_i]$$

- **Generalization of vector quantization**
  - Combine vectors instead of choosing single ones
  - Overcomes limitations due to dictionary sizes
- **Generalization of predefined bases**
  - Dictionary is an overcomplete basis
  - Sparse projection

Finding an optimal dictionary

- **We employ a variation of K-SVD algorithm for dictionary training**
  - Algorithm for designing overcomplete dictionaries for sparse representations [Aharon et al. 06]
  - Alternates sparse coding with fixed dictionary and dictionary update steps
- **But running K-SVD calculations directly on massive volumes would be unfeasible, therefore we reduce training set size**
  - We applied the concept of **coreset** [Agarwal et al. 05] to smartly subsample and reweight the original training set [Feldman & Langberg 11, Feigin et al. 11]
  - We designed a **weighted version of K-SVD**…

Dictionary learning (K-SVD)

- **K-SVD can be seen as a K-Means generalization**

$$\underset{\mathbf{D},\mathbf{\Gamma}}{\text{Min}} \|\mathbf{X}-\mathbf{D}\mathbf{\Gamma}\|_F^2 \qquad \text{Subject To} \quad \forall i \quad \|\underline{\gamma}_i\|_0 \le K$$

- **Basic steps:**
  - Sparse coding of signals in **X**, producing **Γ**
  - Update dictionary atoms given the sparse representations
    - Optimize one atom at a time, keeping the rest fixed

$$\{\underline{d},\underline{g}\} := \underset{\underline{d},g}{\text{Argmin}} \|\mathbf{E}-\underline{d}\,\underline{g}^T\|_F^2 \quad \text{Subject To} \quad \|\underline{d}\|_2 = 1$$

$$\mathbf{E} = \mathbf{X}_I - \sum_{i \neq j} \underline{d}_i \mathbf{\Gamma}_{i,I}$$

    - The size of **E** is proportional to the number of training signals
      - As in [Rubinstein et al. 08] we replace the SVD computation with a simpler numerical approximation

Coreset construction

- **Calculations on massive input volumes are still unfeasible, but we can …**
  - … reduce the amount of data used for training
  - … use importance sampling
- **We associate an importance ι_i to each of the original blocks, being ι_i the standard deviation of the entries in $\mathbf{y}_i$**
  - Picking C elements with probability proportional to $\mathbf{y}_i$
  - Most important blocks should end up in our **coreset**
- **See our 2012 paper for methods for coreset building and training using few streaming passes**

Coreset construction & Weighted K-SVD

- **Non-uniform sampling introduces a severe bias, we thus reweight selected samples**
  - Scale each selected block $\mathbf{y}_i$ by a weight $w_j = \frac{1}{\sqrt{p_j}}$ where p_j is the associated picking probability
- **Applying K-SVD to the non-uniformly scaled training set will converge to a dictionary optimal for the original problem**

$$\sum_j \left\| \tilde{\mathbf{y}}_j - \mathbf{D}\tilde{\boldsymbol{\lambda}}_j \right\|^2 = \sum_j \frac{1}{p_j} \left\| \mathbf{y}_j - \mathbf{D}\boldsymbol{\lambda}_j \right\|^2 \approx \sum_i \left\| \mathbf{y}_i - \mathbf{D}\boldsymbol{\lambda}_i \right\|^2$$

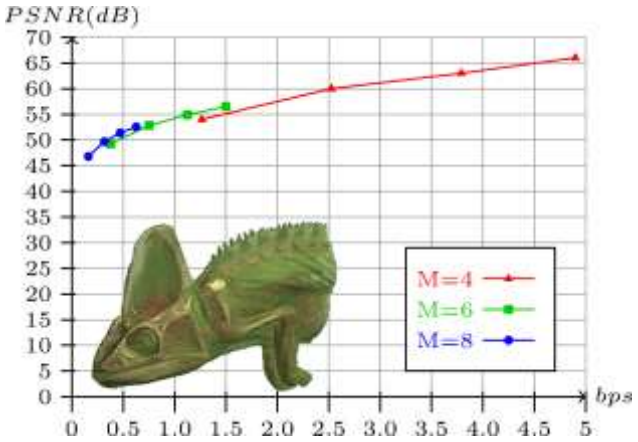
Coreset construction & Weighted K-SVD

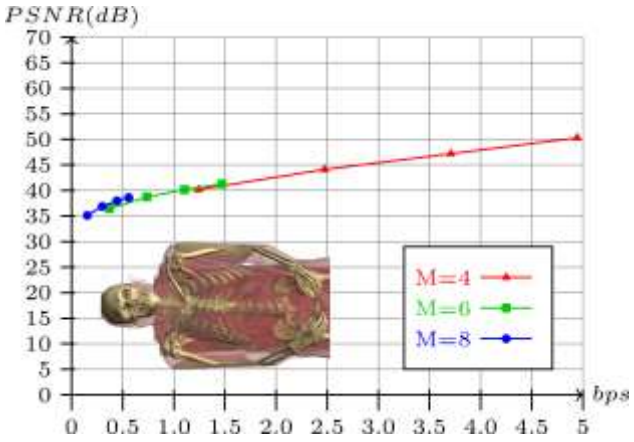
- **Coreset scalability**

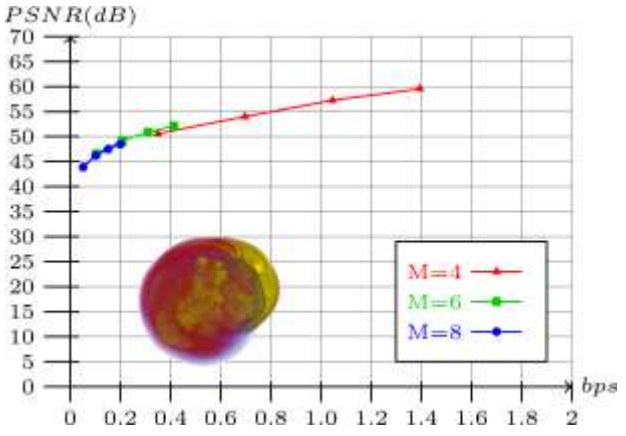
	Chameleon			Visible Human			Supernova		
%	Mvox	Time(h)	PSNR(dB)	Mvox	Time(h)	PSNR(dB)	Mvox	Time(h)	PSNR(dB)
1.4%	18	0.16	52.18	8	0.06	38.19	82	0.76	49.23
3.13%	38	0.29	52.39	16	0.15	38.32	164	1.49	49.30
6.25%	77	0.56	52.57	33	0.28	38.43	329	2.94	49.32
12.50%	154	1.12	52.69	67	0.56	38.54	659	5.79	49.35
25.00%	308	2.21	52.76	134	0.97	38.59	1318	11.73	49.46
50.00%	617	4.37	52.81	268	2.09	38.59	2636	N/A	N/A
100.00%	1234	8.66	52.85	536	4.16	38.60	5273	N/A	N/A

Sparse Coding Results

- **PSNR vs. Bits Per Sample**







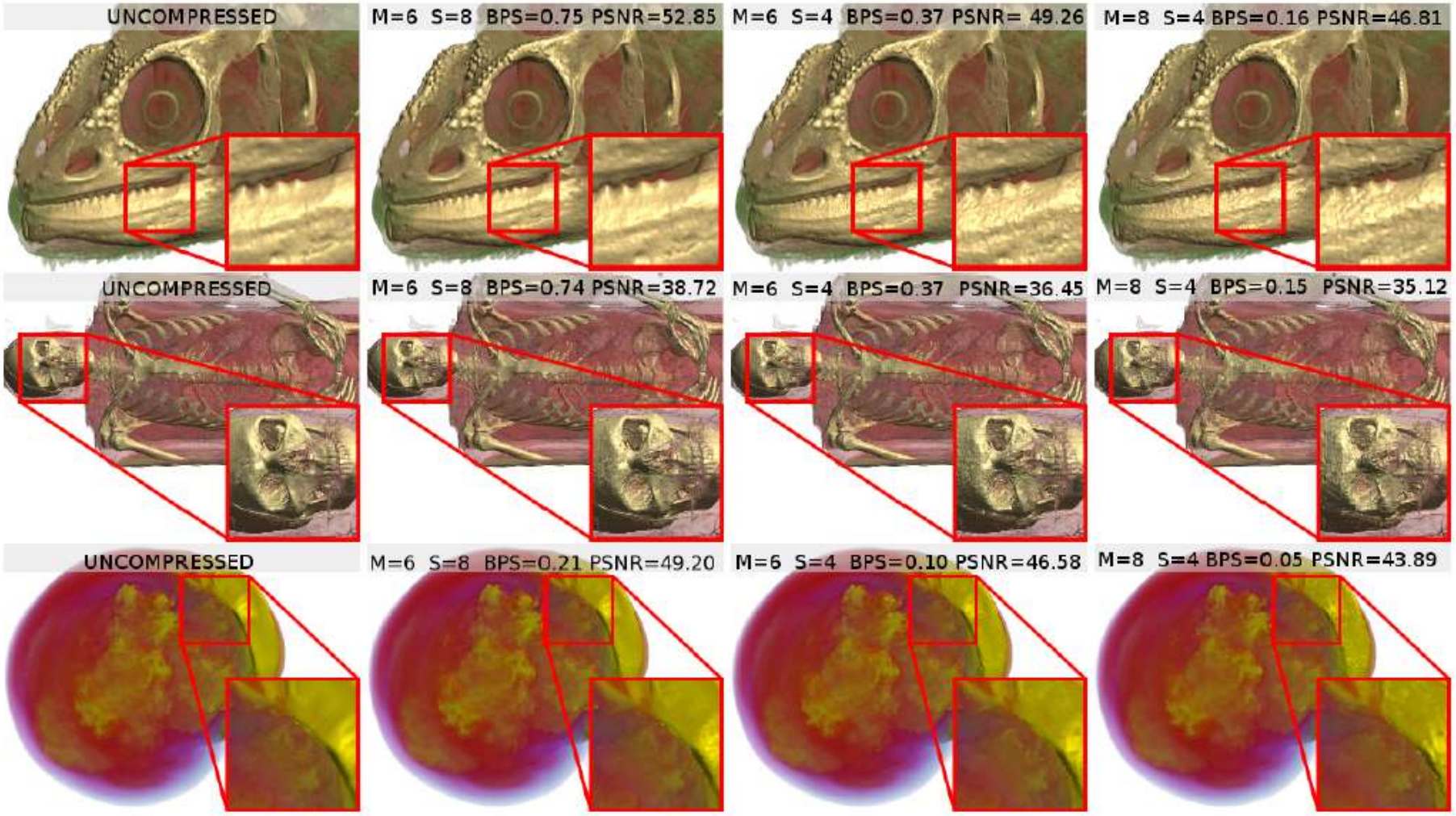
	Chameleon (2.1 GB)				Visible Human (0.91 GB)				Supernova (18 GB)			
K-SVD, $C = 64$Mvox	**M4 S16**	**M6 S8**	**M6 S4**	**M8 S4**	**M4 S16**	**M6 S8**	**M6 S4**	**M8 S4**	**M4 S16**	**M6 S8**	**M6 S4**	**M8 S4**
Size(MB)	1112.1	170.3	85.4	36.4	487.4	72.6	36.5	16.7	1741.3	258.4	129.4	56.6
PSNR(dB)	66	52.44	49.25	46.77	49.74	38.55	36.34	35.02	59.49	49.11	46.57	43.88
Bps	4.9	0.75	0.38	0.16	4.95	0.74	0.37	0.15	1.39	0.21	0.10	0.05
Training time	2h39m	33m	15m	14m	2h40m	32m	14m	12m	2h9m	41m	26m	24m
Encoding time	2h21m	28m	12m	12m	58m	12m	5m	4m	3h13m	45m	23m	21m
Decoding (MVox/sec)	1283	1701	1942	1340	1283	1701	1942	1340	1283	1701	1942	1340

Sparse Coding vs HVQ vs Tensor

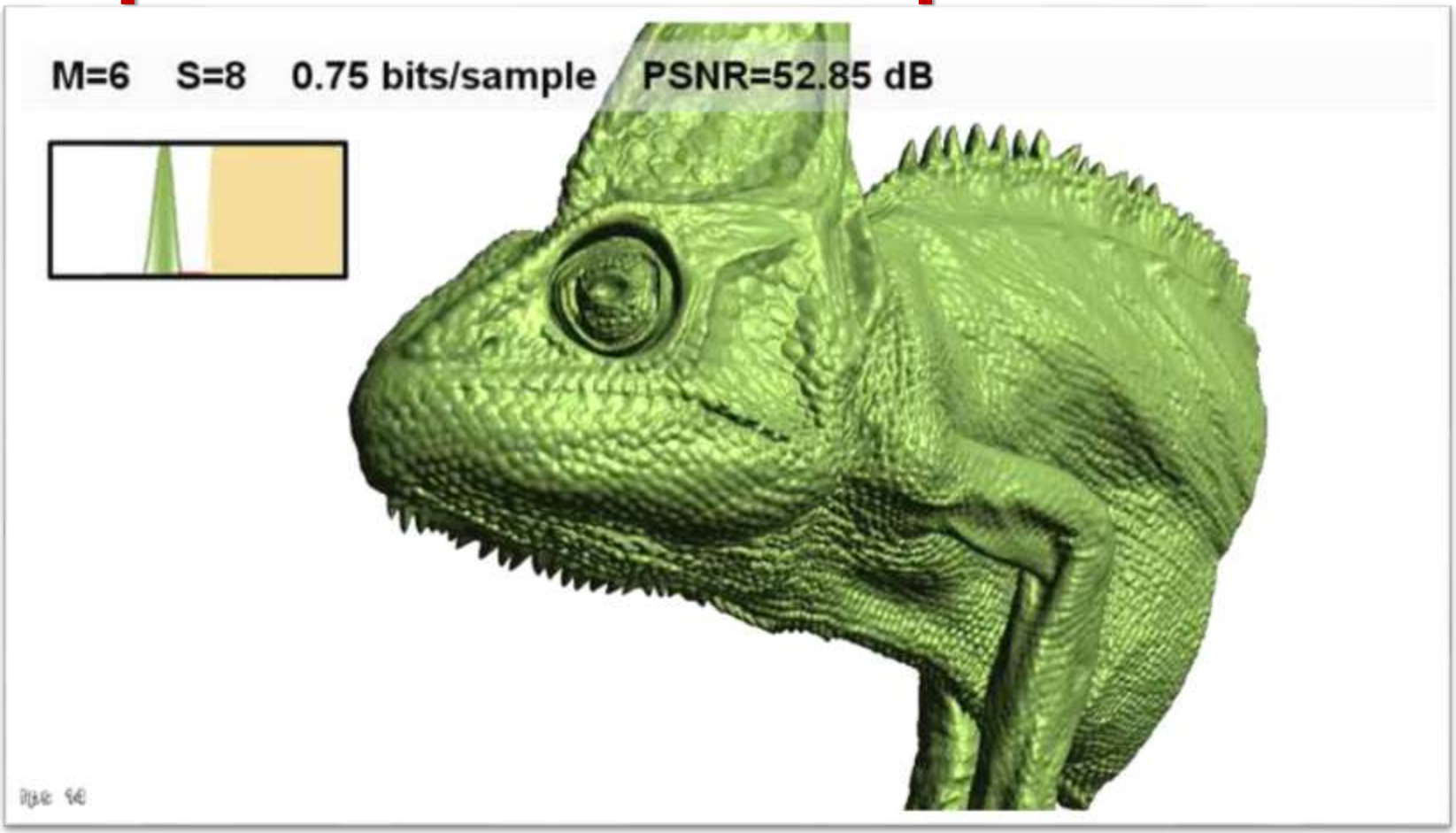
- **Comparison of state-of-the-art GPU-based decompression methods**

		Chameleon (2.1 GB)				Visible Human (0.91 GB)				Supernova (18 GB)			
K-SVD, $C = 64$Mvox, $K = 1024$		M4 S16	M6 S8	M6 S4	M8 S4	M4 S16	M6 S8	M6 S4	M8 S4	M4 S16	M6 S8	M6 S4	M8 S4
SPARSE CODING	Size(MB)	1112.1	170.3	85.4	36.4	487.4	72.6	36.5	16.7	1741.3	258.4	129.4	56.6
	PSNR(dB)	65.98	52.44	49.25	46.77	49.74	38.55	36.34	35.02	59.49	49.11	46.57	43.88
	Bps	4.9	0.75	0.38	0.16	4.95	0.74	0.37	0.15	1.39	0.21	0.10	0.05
	Training time	2h39m	33m	15m	14m	2h40m	32m	14m	12m	2h9m	41m	26m	24m
	Encoding time	2h21m	28m	12m	12m	58m	12m	5m	4m	3h13m	45m	23m	21m
	Decoding (MVox/sec)	1283	1701	1942	1838	1283	1701	1942	1838	1283	1701	1942	1838
HVQ, $C = 64$Mvox		D8192	D4096	D1024	D256	D8192	D4096	D1024	D256	D8192	D4096	D1024	D256
HVQ	Size(MB)	144.5	143.9	143.4	143.3	62.1	61.5	61.1	60.9	218.9	218.3	217.8	217.7
	PSNR(dB)	48.98	48.51	47.53	46.46	37.29	36.84	35.85	34.66	46.93	46.43	45.29	43.75
	Bps	0.64	0.63	0.63	0.63	0.63	0.62	0.62	0.62	0.18	0.17	0.17	0.17
	Training time	1h8m	35m	10m	4m	1h5m	33m	9m	3m	1h19m	46m	23m	17m
	Encoding time	32m	18m	6m	3m	13m	7m	2m	1m	54m	32m	15m	11m
	Decoding (MVox/sec)	2012	2011	2010	2011	2012	2011	2010	2011	2012	2011	2010	2011
TD		R16	R12	R8	R4	R16	R12	R8	R4	R16	R12	R8	R4
TA	Size(MB)	357.2	201.2	102.6	42.2	152.8	86.1	43.9	18.0	502.2	283.0	144.3	59.3
	PSNR(dB)	55.72	46.63	44.56	42.70	39.63	37.68	35.30	31.74	50.60	47.61	43.52	39.42
	Bps	1.48	0.83	0.43	0.17	1.38	0.78	0.40	0.16	0.42	0.24	0.12	0.05
	Encoding time	1h45m	1h22m	1h2m	31m41s	51m	30m	29m	21m	2h22m	1h54m	1h27m	59m
	Decoding (MVox/sec)	415	498	625	781	415	498	625	781	415	498	625	781

Compression Results

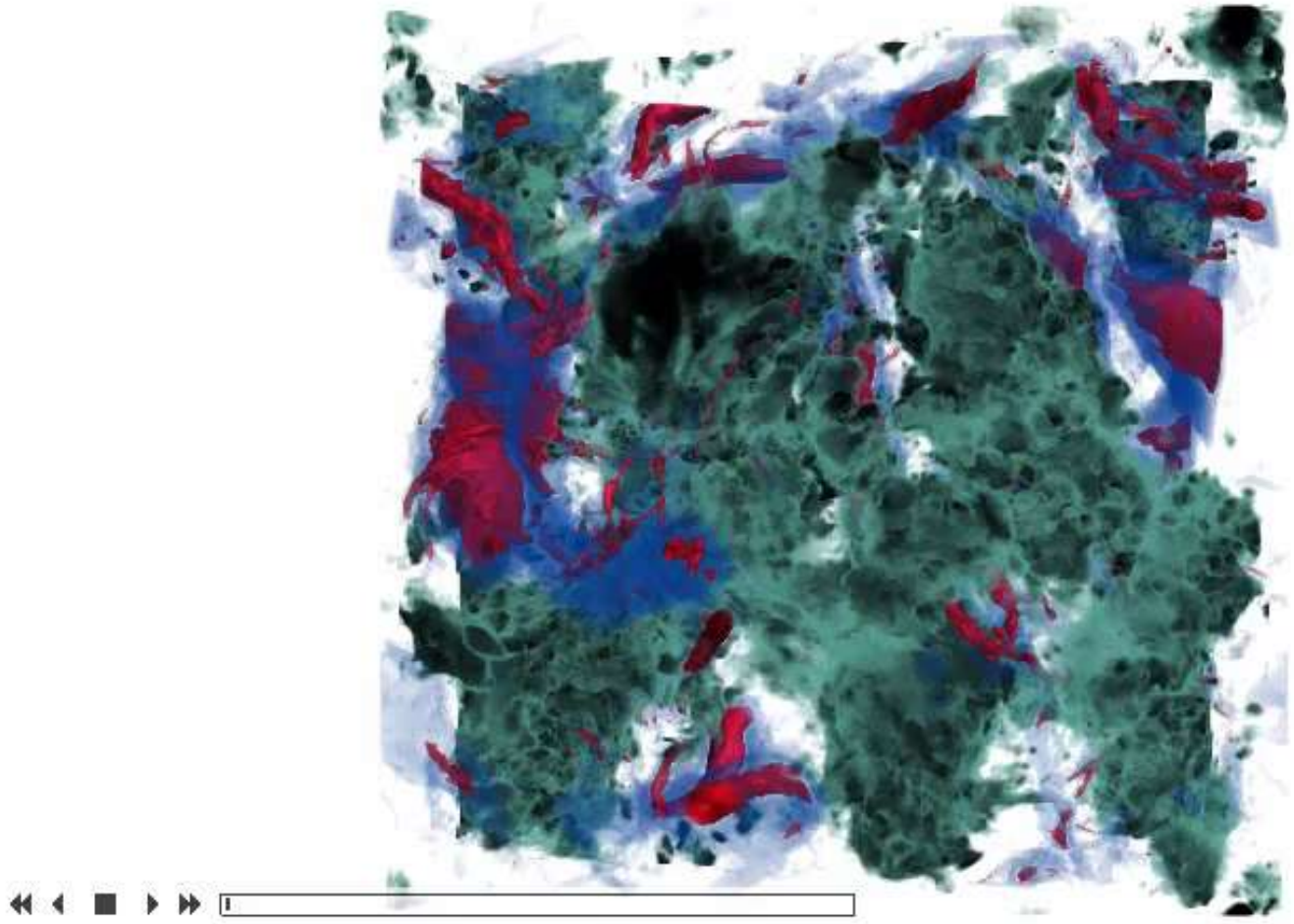


Compressed DVR Example



Chameleon: 1024^3@16bit (2.1GB); Supernova: 432^3×60timesteps@float (18GB)
Compression-domain Rendering from Sparse-Coded Voxel Blocks (EUROVIS2012)

Current work: dynamic data

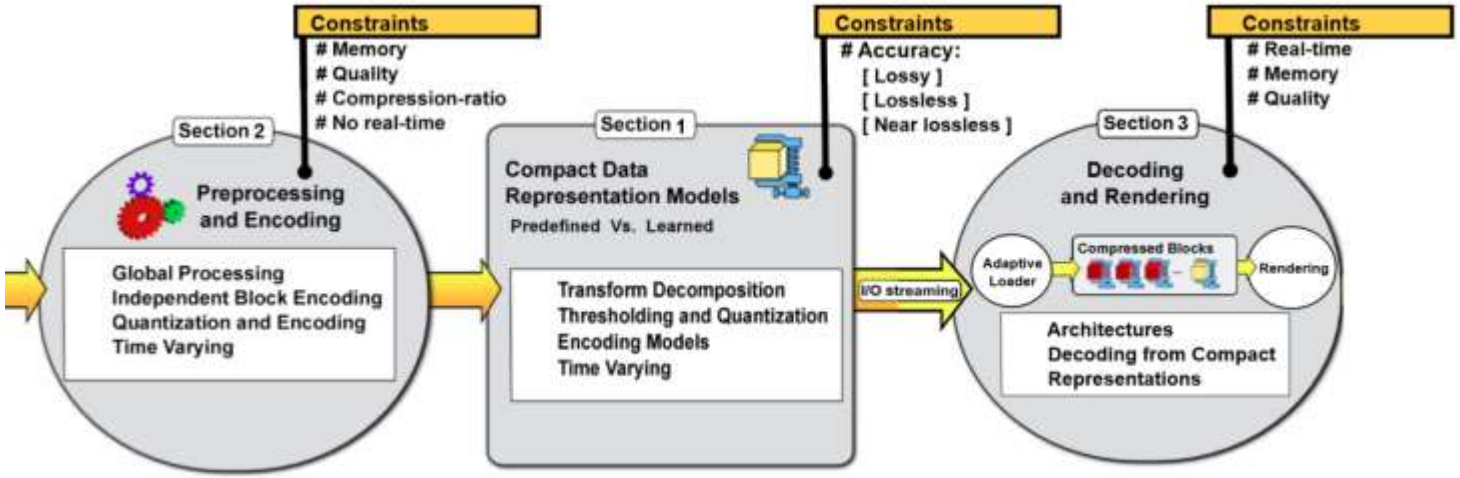


JHU Turbulence 1024^3 x 2048 frames float (8 TB) on Intel i7 16GB RAM with NVIDIA GeForce GTX 980

A bit overtime?

TIME FOR A CONCLUSION, RIGHT?

Summary



- **Provided characterization of the basic components common to modern compressed DVR systems**
  - Compact data models and representation
  - Compression, encoding, decoding and rendering

Ongoing Large Data Problem

- **Generation and acquisition of volume data continues to outperform HW performance capacity**
  - In particular CPU-GPU memory throughput
- **Trend to move all steps to the GPU**
  - Working set selection, decompression & adaptive rendering
  - Aim is just-in-time decompression during rendering to keep data compressed on the GPU
- **Remains an active research area in computer graphics and visualization**
  - Algorithms, data structures & implementation
  - **"Massive volume rendering is one percent inspiration, ninety-nine percent perspiration"** (sort of…)

Limitations & Future Work (Food for PhDs?)

- **Current methods are not fully output-sensitive**
  - Maximum complexity still depends on transparency
  - Budget-based methods exist but are not fully integrated in ray-guided pipelines

Limitations & Future Work (Food for PhDs?)

- **Current GPU accelerated methods have limitations in terms of compression rate vs. quality vs. flexibility**
  - Few of the methods can efficiently cover the full spectrum from extreme compression to (near-)lossless compression
  - Most of current implementations do not adequately support variable bit-rate encoding or error control
  - Blocking artifacts are not completely solved yet
  - Post-process deblocking a bit slow and not flexible

Limitations & Future Work (Food for PhDs?)

- **Not all errors are equal**
  - Errors are measured in data space, typically using Euclidean norms
  - Should work in some perceptual space & define other norms to evaluate error
  - How to take into account transfer function space at preprocessing time while staying flexible?

Limitations & Future Work (Food for PhDs?)

- **Dynamic datasets still pose some unsolved challenges…**
  - Ray-guided streaming & adaptive loading introduces unwanted image variation over time
  - Disturbing for time-varying datasets…
- **… and much more…**

Some documentation

- **Relevant papers**
  - F. Marton, J. A. Iglesias Guitián, J. Diaz, and E. Gobbetti. **Real-time deblocked GPU rendering of compressed volumes**. VMV: 167-174, October 2014.
  - E. Gobbetti, J. A. Iglesias Guitián, and F. Marton. **COVRA: A compression-domain output-sensitive volume rendering architecture based on a sparse representation of voxel blocks**. CGF 31(3-4): 1315-1324, 2012.
  - S. K. Suter, J. A. Iglesias Guitián, F. Marton, M. Agus, A. Elsener, C. Zollikofer, M. Gopi, E. Gobbetti, and R. Pajarola. **Interactive Multiscale Tensor Reconstruction for Multiresolution Volume Visualization.** IEEE TVCG, 2011.
  - M. Agus, E. Gobbetti, J. A. Iglesias Guitián, and F. Marton. **Split-Voxel: A Simple Discontinuity-Preserving Voxel Representation for Volume Rendering.** Volume Graphics: 21-28, 2010.
  - J. A. Iglesias Guitián, E. Gobbetti, and F. Marton. **View-dependent Exploration of Massive Volumetric Models on Large Scale Light Field Displays**. The Visual Computer, 26(6-8): 1037-1047, 2010.
  - E. Gobbetti, F. Marton, and J. A. Iglesias Guitián. **A single-pass GPU ray casting framework for interactive out-of-core rendering of massive volumetric datasets**. The Visual Computer, 24(7-9): 797-806, 2008.
- **Recent surveys**
  - M. Balsa Rodriguez, E. Gobbetti, J. A. Iglesias Guitián, M. Makhinya, F. Marton, R. Pajarola, and S. Suter. **State-of-the-art in Compressed GPU-Based Direct Volume Rendering**. CGF, 33(6): 77-100, September 2014.
  - J. Beyer, M. Hadwiger, and H. Pfister, **State-of-the-Art in GPU-Based Large-Scale Volume Visualization**. Computer Graphics Forum. 2015. To appear.

Contacts:

gobbetti@crs4.it

http://www.crs4.it/vic

Work partially supported by:
DIVA - Data Intensive Visualization and Analysis
EU ITN FP7/2007-2013/
REA grant agreement 290227

That's all Folks!

©Leo Ortolani, Rat-Man